



The Type System of VCL Structural and Assertion Diagrams

Nuno Amálio
Laboratory for Advanced Software Systems
University of Luxembourg
6, rue R. Coudenhove-Kalergi
L-1359 Luxembourg

TR-LASSY-11-04

Version	Date	Description
0.1	22/04/2011	1st release with type system of structural and assertion diagrams
0.2	23/03/2012	Updated type system of structural and assertion diagrams
0.3	12/06/2012	Updated with section and proofs of information preserving isomorphism between metamodels and grammars.
0.4	15/02/2012	Updated diagrams to meet new concrete syntax. Added Quantifiers.

Table 1: Document Revision History

Contents

1	Introduction	5
1.1	Background: The Visual Contract Language (VCL)	5
1.1.1	VCL Diagrams	5
1.1.2	VCL Syntax and Semantics	5
1.2	Outline	6
2	Running Example	7
3	Syntax	10
3.1	Metamodels	10
3.1.1	Common	10
3.1.2	Structural Diagrams	12
3.1.3	Assertion Diagrams	13
3.2	Grammars	15
4	From Metamodels to Grammars and Back	17
4.1	Overall setting	17
4.2	VCL Syntactic Isomorphisms	17
4.2.1	Isomorphism Theorems and their Proofs	18
	Proofs for the common part	18
	Proofs for the SD part	18
	Proofs for the AD part	19
5	Type System	20
5.1	Types and Environments	20
5.2	Base Rules	21
5.3	Common Rules	22
5.4	Rules for Structural Diagrams	27
5.5	Rules for Assertion Diagrams	31
A	Auxiliary Definitions	36
A.1	Environment Operators	36
A.2	Predicates	37
A.3	Auxiliary Functions	37
A.3.1	Function <i>getGType</i>	37
A.3.2	Functions producing variable environments (VEs)	38
A.3.3	Function <i>getDK</i>	38
A.3.4	Functions to extract information from ADs	38

A.3.5	Functions for AD lookup	39
A.3.6	Functions for substitutions	39
A.3.7	Function <i>getSidFrScalarOrCollection</i>	40
B	Alloy Metamodels	41
B.1	VCL Common	41
B.2	Bool Module	50
B.3	VCL Structural Diagrams	51
B.4	VCL Assertion Diagrams	56
C	Z3 Proofs	66
C.1	Common	66
C.1.1	Z3 Encoding	66
C.1.2	Z3 Proof Output	93
C.2	Structural diagrams	94
C.2.1	Z3 Output	110
C.3	Assertion diagrams	111
C.3.1	Z3 Output	133

Chapter 1

Introduction

This document presents a type system for the Visual Contract Language (VCL) [AK10, AKMG10], covering structural and assertion diagrams. This formalises a typed object-oriented system with subtyping. This type system has been implemented in the VCL tool, the *Visual Contract Builder*¹ [AGK11]. The following gives some background on VCL and an outline of the overall document.

1.1 Background: The Visual Contract Language (VCL)

VCL [AK10, AKMG10, AGK11] is a formal language for the abstract modelling of software designs. Its modelling paradigms are set theory, object-orientation and design-by-contract (pre- and post-conditions). VCL's distinguishing features are its capacity to describe predicates visually and its approach to behavioural modelling based on design by contract.

VCL's semantics is based on set theory. Its semantics definition takes a translational approach. Currently, VCL has a Z semantics: VCL diagrams are mapped to ZOO [APS05, Amá07], a semantic domain of object orientation for the language Z [Spi92, ISO02].

1.1.1 VCL Diagrams

A VCL model is made up of diagrams of different kinds. VCL's diagram suite comprises: *package*, *structural*, *behaviour*, *assertion* and *contract* diagrams. Package diagrams (PDs) define VCL packages, coarse-grained modules, and their dependencies with other packages. Structural diagrams (SDs) define state structures and their relations that together make the state space of a package (e.g. Fig. 2.1a). Behaviour diagrams (BDs) provide a map over the behaviour units of a package. ADs define predicates over a single state, which are used to define invariants and query operations (e.g. Figs. 2.1b to 2.1i). Finally, contract diagrams (CDs) describe operations that change state through a contract (a pre- and a post-condition). The type system presented here covers SDs and ADs only.

1.1.2 VCL Syntax and Semantics

VCL's semantic domain is detailed in [APS05]. Briefly, syntax and semantics of SDs and ADs are as follows:

¹<http://vcl.gforge.uni.lu/>

- All rounded contours in Fig. 2.1a are *sets* (or *blobs*). *Objects* are represented as labelled rectangles; they are atoms, members of a set of possible objects.
- In a SD, a set can either be *value* or *class*. In Fig. 2.1a, **Customer** and **Account** are classes, and all others are value sets. Value sets represent values; class sets (like OO classes) represent objects with identity.
- Property edges are represented as directed arrows. In a SD, property edges define properties shared by all objects of a set (e.g. **custNo**, **accNo** and **balance** in Fig. 2.1a)). In ADs, property edges are used to state predicates that relate the source set or object with some target expression.
- Relation edges are labelled directed lines; direction is indicated by arrow symbol above the line (e.g. **Holds** in Fig. 2.1a).
- SDs define state spaces. ADs describe assertions (conditions or predicates) on a state space. A global (or package) state is a collection of object states, together with states of relation edges. Object states are functions that map object identifiers to their states; there is such a function for each class set. Semantically, a relation edge is a binary relation; it denotes a set of tuples.

1.2 Outline

The remainder of this document is as follows:

- Chapter 2 presents the running example that is used to illustrate the type system presented here.
- Chapter 3 presents the syntactic descriptions of VCL structural and assertion diagrams, from which the type system is defined.
- Chapter 4 discusses the mapping from metamodels to grammars for the purpose of defining the type system, showing that this mapping is sound.
- Chapter 5 presents the actual type system of VCL structural and assertion diagrams.
- Appendix A presents the auxiliary definitions that are used to describe VCL's type system presented here.
- Appendix B presents the VCL metamodels describe using the Alloy formal modelling language.
- Appendix C presents the Z3 encodings for the graphs of metamodels and grammars together with the results of the isomorphism proofs.

Chapter 2

Running Example

This paper's running example is the *Simple Bank* case study [AK10]¹. Figure 2.1 give several diagrams of this case study's VCL model. The SD (Fig. 2.1a) is as follows:

- The two class sets, **Customer** and **Account**, represent, respectively, bank customers and bank accounts.
- Value sets **CustId**, **Name** and **Address** represent, respectively, sets of identifiers, names and addresses of bank customers. **CustType** defines the possible types of customers (a definitional set, symbol \bigcirc): constant objects **corporate** and **personal**. **AccID** represents set of account identifiers. **Int** (a primitive set) represents the integers. **AccType** (a definitional set) represents the possible kinds of accounts: constant objects **savings** and **current**.
- Relation-edge **Holds** relates customers and their accounts. Assertions (elongated hexagons) identify invariants, which can either be *local* (linked to a set) or *global* (not linked).

Local **Account** invariant **SavingsArePositive** (Fig. 2.1b) says, using an implication formula, that savings accounts must be positive. This AD results in the Z predicate: $aType = savings \Rightarrow balance \geq 0$. The same invariant is described globally using a set formula in Fig. 2.1c; this says that the set of negative savings accounts (inner set) must be empty (shading). This results in the Z predicate:

$$\{o : sAccount \mid (stAccount\ o).aType = savings \wedge (stAccount\ o).balance < 0\} = \emptyset$$

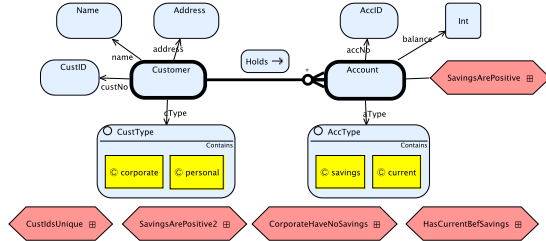
$sAccount$ is set of all existing account objects; $stAccount$ is a function mapping account objects to their states.

Global invariant **CustIdsUnique** (Fig. 2.1d) says that customer identifiers are unique. The AD says this using a quantifier formula: for all pairs of distinct customer objects, their customer numbers must also be distinct. This results in the Z predicate:

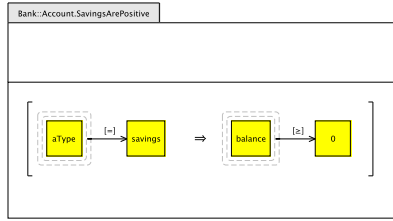
$$\forall c1, c2 : sCustomer \bullet c1 \neq c2 \Rightarrow (stCustomer\ c1).custNo \neq (stCustomer\ c2).custNo$$

Global invariant **CorporateHaveNoSavings** (Fig. 2.1e) says that corporate customers cannot have savings accounts. The AD builds a set by restricting relation **Holds** using property edge modifiers (edges with double-arrow): the domain is restricted to the set of corporate customers

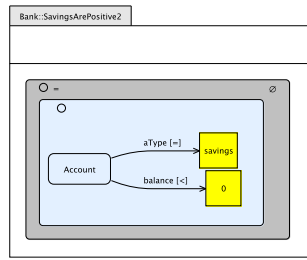
¹A tutorial using this case study is available at <http://vcl.gforge.uni.lu/SBDemo>.



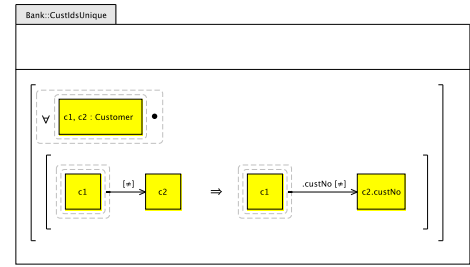
(a) Structural Diagram



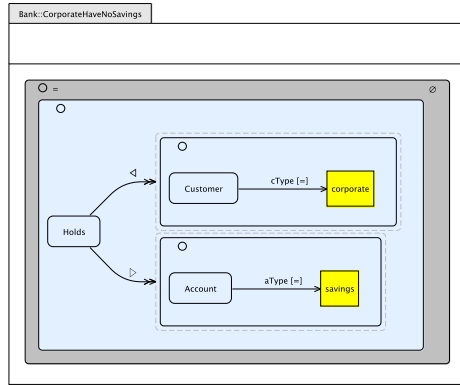
(b) A local invariant



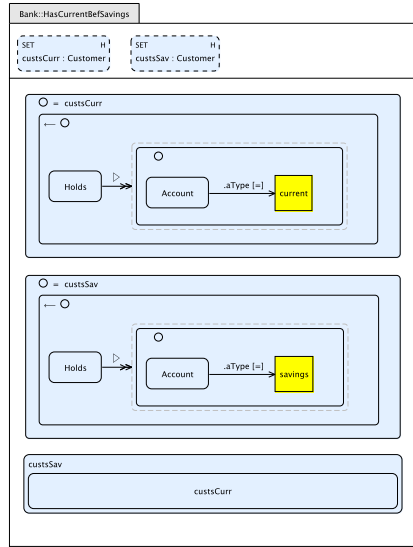
(c) A global invariant



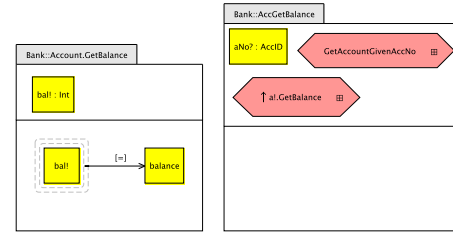
(d) A global invariant



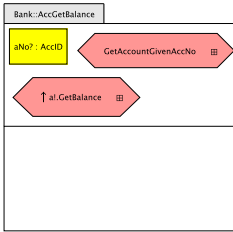
(e) A global invariant



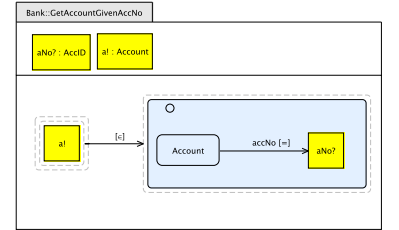
(f) A global invariant



(g) A local Operation



(h) A global Operation



(i) A global Operation

Figure 2.1: Sample assertion diagrams of the *simple bank* VCL model

(symbol \triangleleft); the range to the set of savings accounts (symbol \triangleright). The outer set is shaded to say that this constructed set must be empty. The resulting Z is:

$$(\{o : sCustomer \mid (stCustomer\ o).cType = corporate\} \triangleleft rHolds) \triangleright \{o : sAccount \mid (stAccount\ o).aType = savings\} = \emptyset$$

Here, \triangleleft and \triangleright are domain and range restriction relation operators.

Global invariant **HasCurrentBefSavings** (Fig. 2.1f) says that customers must have a current account before opening a savings account using a subset formula. This involves building two auxiliary sets (respectively): (a) set of customers with current accounts (local or hidden variable **custsCurr**) and (b) set of customers with savings accounts (local or hidden variable **custsSav**). Both sets are built similarly by taking the domain (symbol \leftarrow) of **Holds** restricted on the range. AD of Fig. 2.1f imports the auxiliary ADs (represented as assertions) and says that **custsSav** is a subset of **custsCurr**; as **custsSav** and **custsCurr** are not declared in Fig. 2.1f, they are internal variables hidden to the outside world. The Z resulting from Fig. 2.1f is:

$ \begin{array}{l} \text{BankHasCurrentBefSavings0} \\ \text{BankGblSt} \\ \text{custsCurr} : \mathbb{P} \mathbb{O} \text{ CustomerCl} \\ \text{custsSav} : \mathbb{P} \mathbb{O} \text{ CustomerCl} \\ \hline \text{custsCurr} = \text{dom}(rHolds \triangleright \{o : sAccount \mid (stAccount\ o).aType = current\}) \\ \text{custsSav} = \text{dom}(rHolds \triangleright \{o : sAccount \mid (stAccount\ o).aType = savings\}) \\ \text{custsSav} \subseteq \text{custsCurr} \end{array} $

$$\text{BankHasCurrentBefSavings} == \text{BankHasCurrentBefSavings0} \setminus (\text{custsCurr}, \text{custsSav})$$

Above, variables **custsSav** and **custsCurr** are hidden using the \setminus Z operator.

In VCL, queries are defined using ADs, which can be local or global. Often, global operations are built from local ones. Local operation **Account.GetBalance** (AD of Fig. 2.1g) retrieves the balance of some account object and stores it in output variable **bal!**. Global operation **AccGetBalance** (Fig. 2.1h) retrieves some account balance given some account number (input **aNo?**); this involves obtaining the object account (**a!**) associated with **aNo?** through operation **GetAccount-GivenAccNo** (Fig. 2.1i) and then retrieving the account's balance using **Account.GetBalance**. The \uparrow symbol says that variables, and not only the predicate, are imported; this means that output **bal!** is defined also in **AccGetBalance**.

This running example highlights the utility of typing. For instance, in AD of Fig. 2.1e it would be useful to check that **Holds**, **Customer** and **Account** are sets defined in SD of Fig. 2.1a, that the operators \triangleleft and \triangleright are applied correctly, and that the properties **cType** and **aType** exist and are applied correctly with respect to the operator $=$. Similarly for the remaining ADs.

Chapter 3

Syntax

This chapter presents the syntax of VCL structural and assertion diagrams in terms grammars and class metamodels. The metamodels are the primary representation; metamodels are the basis for constructing the graphical parsers of VCL's tool. The grammars are used to describe the type system; in the implementation the grammar representation is used for type-checking and translation to Z.

3.1 Metamodels

The metamodels of the VCL notations presented here have been defined in the Alloy specification language [Jac06]. They are given in appendix B. Here, we present these metamodels using UML class diagrams, which partially describe what is described in Alloy: the Alloy describes constraints that are not describable using class diagrams.

The Alloy metamodels of VCL package, structural and assertion diagrams comprises the following modules: *common* (section B.1), *structural diagrams* (section B.3) and *assertion diagrams* (section B.4). The following class diagrams describe each of these modules.

3.1.1 Common

The metamodel of the part that is common to both SDs and ADs (Fig. 3.1), corresponding to the Alloy module of section B.1, is as follows:

- Several constructions have a name attribute; the metaclass (**Name**, bottom-left) denotes all names of a VCL model. Several constructions use the type designator (**TypeDesignator**, bottom-left). A type designator can either denote the set of natural numbers (**TypeDesignatorNat**), the set of integers (**TypeDesignatorInt**), or some set defined by a blob or relation edge and denoted by their identifier (**TypeDesignatorId**).
- A property edge (**PropEdge**) can either be of type *predicate* (**PropEdgePred**) or *modifier* (**PropEdgeMod**). **PropEdgePreds** comprise a unary and binary operator (**uop** and **bop** association-ends), an instance of **EdgeOperatorUn** and **EdgeOperatorBin**, respectively, a target **Expression** (**target** association-end) and an optional designator (attribute **designator**) to refer to some property of a blob. A **PropEdgeMod** comprises a modifier operator (**mop** association-end) an instance of **EdgeOperatorMod**.

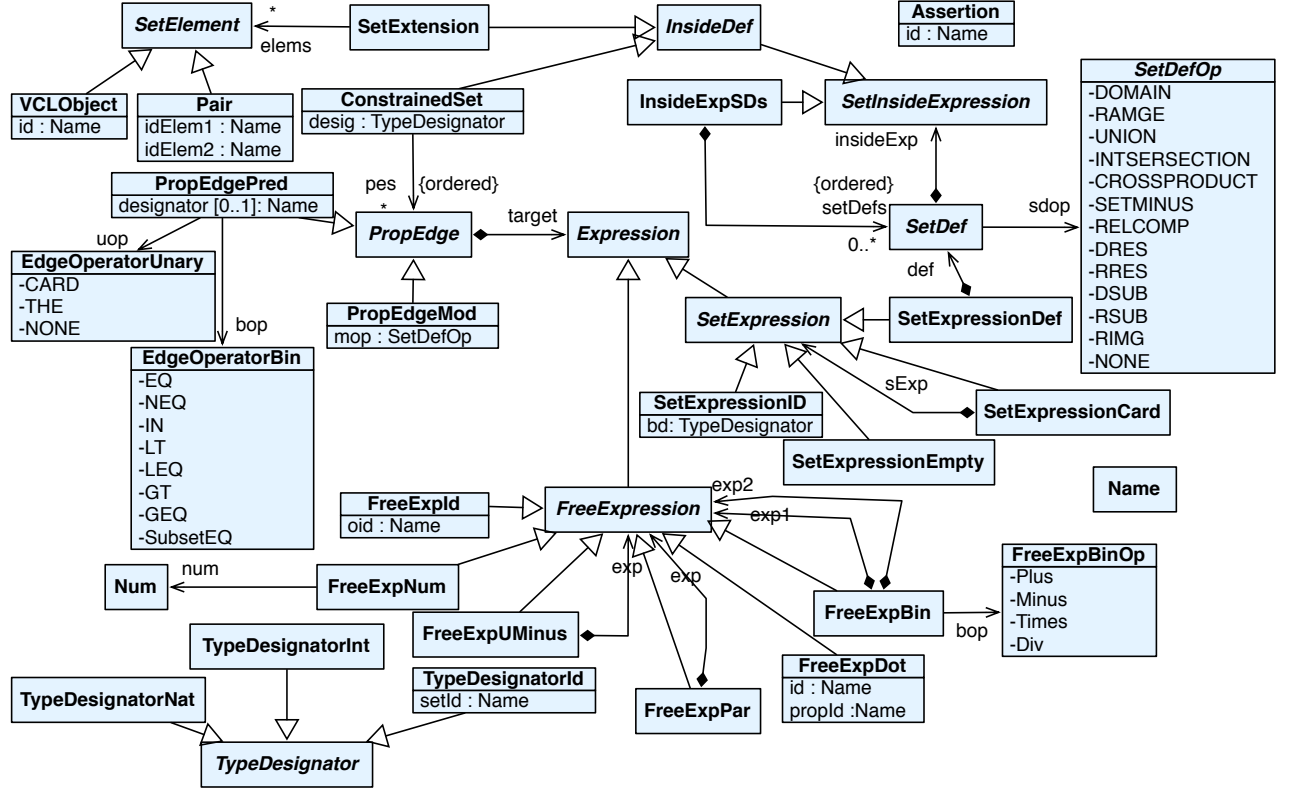


Figure 3.1: The common metamodel

- A modifier edge operator (**EdgeOperatorMod**) is an enumeration comprising the operators: domain restriction (**DRES**, \triangleleft), range restriction (**RRES**, \triangleright), domain subtraction (**DSUB**, \boxminus), range subtraction (**RSUB**, \boxtimes) and relation image (**RIMG**, $\llbracket \rrbracket$). A predicate edge operator is enumeration comprising the operators: equality (**EQ**, $=$), difference (**NEQ**, \neq), set membership (**IN**, \in), less then (**LT**, $<$), less or equal then (**LEQ**, \leq), greater then (**GT**, $>$), greater or equal then (**GEQ**, \geq), and subsetting (**SubsetEQ**, \subseteq).
- There are two kinds of expressions: *object* (**ObjExpression**), represented as objects (rectangles), and *set* (**SetExpression**), represented as blobs (rectangles with rounded corners). An object expression can either be: an identifier (**ObjExpId**); a number (**ObjExpNum**); a unary minus expression (**ObjExpUMinus**), comprising another expression (**exp** association-end); a binary object expression, comprising two expressions (association-ends **exp1** and **exp2**) and an infix operator (**bop** association-end); or a parenthesised expression, comprising another expression (**exp** association-end). A binary object-expression operator (**ObjExpBinOp**) is an enumeration comprising the operators: **Plus** (+), **Minus** (-), **Times** (*), and **Div** (div).
- A **SetExpression** can either refer to some existing set (**SetExpressionID**), denote the empty set (a blob that is shaded), be a cardinality operator applied to another set expression **SetExpressionCard**, or be a set definition (**SetExpressionDef**). A **SetExpressionID** comprises a designator of the set being referred (attribute **design**). A **SetExpressionCard**

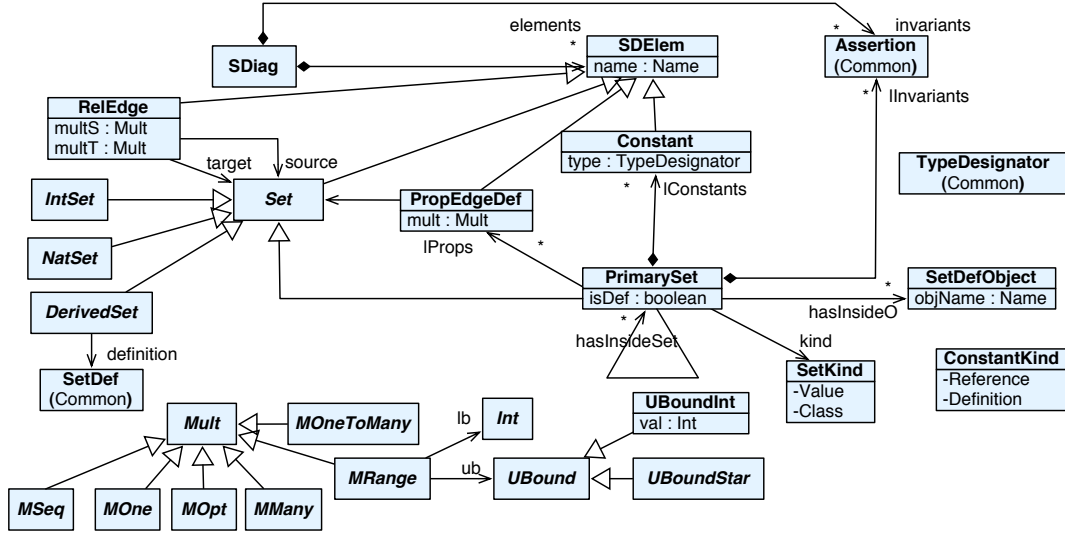


Figure 3.2: The metamodel of VCL Structural diagrams

is the cardinality operator applied to another set expression (**sExp** association-end). A **SetExpressionDef** comprises a set definition (association-end **def**), an instance of **SetDef**.

- Set definitions (**SetDef**) are defined by the things they have inside. They comprise an inside expression (**insideExp** association-end), representing the expression placed inside the blob, and by a set definition operator (**sdop** association-end). A set definition operator (**SetDefOp**) is an enumeration defining the operators **Domain** (symbol \leftarrow), **Range** (symbol \rightarrow), **Union** (symbol \cup), **Intersection** (symbol \cap), **CrossProduct** (symbol \times), **SetMinus** (symbol \setminus) or **None** (no operator).
- A set inside expression (**SetInsideExpression**) is either an inside definition (**InsideDef**) or a sequence of set definitions (**InsideExpSDs**). A **InsideExpSDs** comprises a sequence of set definitions (**setDefs** association-end). An **InsideDef** is an abstract class, which comprises either a **SetExtension** or a **ConstrainedSet**. A **ConstrainedSet** represents a set constrained with an ordered collection of property edges (association-end **pes**). A set extension (**SetExtension**) represents a set defined extensionally by a set of elements (association-end **elems**), which are instances of **SetElem**.
- A **SetElem** is represented visually as a rectangles; it can either be a **VCLObject** (a member of set) or a **Pair** (a member of a relation). A **VCLObject** comprises a name (the name of the object); a **Pair** comprises a pair of names.

3.1.2 Structural Diagrams

The metamodel of VCL structural diagrams (SDs) (Fig. 3.2), corresponding to the Alloy module of section B.3, is as follows:

- A SD (**SDDiag**) is made of structural elements (**SDElem**) and invariants (**Assertion**). A **SDElem** can be a relation edge (**RelEdge**), constant (**Constant**) or set (**Set**).

- In a SD, an **Assertion** represents an invariant. If they belong to the overall SD (association-end **invariants**) they represent global invariants; if they are connected to a set (association-end **lInvariants**), the invariant is local to the set.
- A relation edge (**RelEdge**), or association, represents an edge between two sets: the **source** and the **target**. It holds two attributes to record the multiplicities attached to source and target (**multS** and **multT**).
- Like invariants, constants (**Constant**) are global if they are not connected to any set and local otherwise (association-end **lConstants**).
- A set can be primary (**PrimarySet**), derived (**DerivedSet**) or one of the sets corresponding to primitive types: integers (**IntSet**) or natural numbers (**NatSet**).
- A derived set has a name (attribute **id**) and is associated with a set definition (**SetDef**, defined in common metamodel).
- A primary set has a kind (**SetKind**), indicating whether the set is **Class** or **Value**. A primary set comprises a set of local constants (association-end **lConstants**), a set of local invariants (association-end **lInvariants**), and a set of property edge definitions (association-end **lProps**). A primary set may have other primary sets and objects inside (association-ends **hasInsideSet** and **hasInsideO**).
- A property edge definition (**PropEdgeDef**) has a set has the edge's target (association-end **peTarget**) indicating the type of the property, and a multiplicity constraint (attribute **mult**).
- Multiplicities (**Mult**) are attached to relation edges and property edge definitions. A multiplicity can either be single (**MOne**), optional (**MOpt**), sequence (**MSeq**), multiple with 0 or more (**Many**), multiple with at least one (**MOneToMany**), or be defined as a range (**MRange**) comprising a lower and an upper bound (association-ends **ub** and **lb**).

3.1.3 Assertion Diagrams

The metamodel of VCL assertion diagrams (Fig. 3.3), corresponding to the Alloy module of section B.4, is as follows:

- An assertion diagram (**ADiag**) comprises a name (**aName**), a set of declarations corresponding to the declarations compartment (**declarations** association-end), and a set of formulas corresponding to the predicate compartment (**predicate** association-end).
- A declaration (**Decl**) can either be a typed declaration (**TypedDecl**) or a declaration formula (**DeclFormula**). A typed declaration has a name (**dName**) and a type (**dTy**), and it can either be a declaration of an object (**DeclObj**) or the declaration of a set (**DeclSet**). The **sequence** attribute of **DeclSet** indicates whether the set is a normal set (value **false**) or a sequence (value **true**). The **optional** attribute of **DeclObj** indicates whether the object is optional or not.
- A formula (**Formula**) can either be a negation formula (**FormulaNot**), a binary formula (**FormulaBin**), an arrows formula (**ArrowsFormula**) or a set formula (**SetFormula**).

- A declarations formula (**DeclFormula**) can either be a declarations formula atom (**DeclFormulaAtom**), which comprises a declaration reference, a negated declaration formula (**DeclFormulaNot**), which comprises the declarations formula being negated, or a binary declaration formula (**DeclFormulaBin**), which comprises an operator (**DeclFormulaBinOp**) and two declarations formulas.
- **FormulaSource** represents the source of a predicate formula. This source can either be a set element (**FormulaSourceElem**), which comprises a **SetElement** (defined in **Common**, Fig. 3.1), a set (**FormulaSourceSet**) or a be some unary operator applied to a formula source **FormulaSourceUnary**.

3.2 Grammars

The following presents the grammars of VCL structural and assertion diagrams; they are equivalent to the visual metamodels presented above.

The grammars use the following operators:

- \bar{x} for zero or more repetitions of x ;
- \bar{x}^1 for one or more repetitions of x ;
- $x \mid y$ for a choice of x or y ;
- $[x]$ for an optional x .

In addition,

- $\bar{x}c$ for some character symbol c means zero or more occurrences of x separated with c ;
- $\bar{x}c^1$ for some character symbol c means one or more occurrences of x separated with c ;

Symbols are set in bold type when they are to be interpreted as terminals to avoid confusion with grammar symbols. We introduce two syntactic sets, representing terminals: the set of identifiers *Id*, and the set of numeric constants (*Num*).

TD ::= **Int** | **Nat** | **Id**
 O ::= **object** [**Id** | **self**]
 P ::= **pair** ([**Id** | **self**], [**Id** | **self**])
 SE ::= O | P
 A ::= **assertion** **Id**
 PE ::= (PEP | PEM) TExp
 PEP ::= UEOp [**Id**] BEOp
 UEOp ::= # | • | **elems** | ϵ
 BEOp ::= = | \neq | \in | < | \leq
 | > | \geq | \subseteq
 PEM ::= [SOp] \Rightarrow
 TExp ::= FExp | SExp
 FExp ::= **Id** | **self** | [**Id** | **self**].**Id**
 | Num | -FExp
 | FExp FEOp FExp
 | (FExp)
 FEOp ::= + | - | * | **div**
 SExp ::= **set** TD | **set iden** | SDef
 | **set shaded**
 | UEOp SExp
 SDef ::= **set** \bigcirc SOp **hasIn** {IExp}
 SOp ::= \leftarrow | \rightarrow | \cap | \cup | \times | \setminus | \boxtimes
 | \triangleleft | \triangleright | \boxtimes | \boxtimes | $\llbracket \rrbracket$ | \perp
 IExp ::= IDef | \overline{SDef} ;
 IDef ::= **set** TD { \overline{PE}^1 } | \overline{SE}^1

(a) Common Syntax

SD ::= **SD STRUCTURES:** \overline{SDE} **INVARIANTS:** \overline{A}
 SDE ::= C | RE | Set
 C ::= **const** **Id** : TD
 M ::= **opt** | **one** | **some** | **many**
 | **seq** | Num .. (Num | *)
 RE ::= **relEdge** **Id** (M TD, M TD)
 SK ::= **value** | **class**
 Set ::= PSet | **Id** \leftrightarrow SDef
 PSet ::= **set** **Id** SK [\bigcirc] { $\overline{C} \overline{PED} \overline{A}$ }
 | **hasIn** { (\overline{O} | \overline{PSet}) }
 PED ::= **Id** \rightarrow M TD

(b) Structural Diagrams

AD ::= **AD** **Id** [:**Id**] **DECLARATIONS:** \overline{D} **PREDICATE:** \overline{F}
 D ::= VD | DF
 VD ::= [**hidden**] DV \overline{Id} , : TD
 DV ::= [**opt**] **object** | **set** | **seq**
 R ::= **Id** / **Id**
 DFA ::= [\uparrow] **assertion** [**Id** \rightarrow] [**Id** .] **Id** [\overline{R}]
 DF ::= DFA | FOp [\overline{DF}]
 FOp ::= \Rightarrow | \Leftrightarrow | \wedge | \vee | \neg | \boxtimes
 F ::= AF | SF | FOp [\overline{F}] | QF
 AFS ::= SE | AFSS | FSOp AFS
 AFSS ::= **set** **Id** | SDef
 FSOp ::= # | \leftarrow | \rightarrow | •
 AF ::= AFS { \overline{PEP} }
 SF ::= [**shaded**] [**Id**] SDef | **set shaded** TD
 | **set** TD **hasIn** {SExp}
 QF ::= \overline{QD} , • \overline{F} ;
 QD ::= (\forall | \exists) \overline{VD} ;

(c) Assertion Diagrams

Figure 3.4: Syntax of VCL Structural and Assertion diagrams

Chapter 4

From Metamodels to Grammars and Back

This chapter demonstrates that metamodel and grammar representations of chapter 3 are equivalent, which means that it is straightforward to go from one representation to the other. This is important because the type system presented in the next chapter is defined on the grammar, but the graphical editors of VCL's tool are based on metamodels. This chapter shows that this approach based on these two representations is sound.

4.1 Overall setting

In [EEPT06], a graph is defined as a tuple $G = \langle V, E, s, t \rangle$, where V is a set of nodes, E is a set of edges, and $s, t : E \rightarrow V$ are the source and target functions, respectively, assigning to each edge a source and a target node. A metamodel is actually a typed graph [EEPT06], but this does not concern us here. We are interested in going from the metamodel to the grammar.

A grammar (in our context, a context-free grammar) is defined as the tuple $Gr = \langle V, \Sigma, S, P \rangle$, where V is a set of non-terminals, Σ a set of terminals, S is the starting symbol (it is a member of V) and P is a set of grammar rules or productions. The abstract syntax induced by a grammar can be represented as a graph (a special kind of graph, a tree), where the nodes are the terminals and non-terminals of the grammar, the root node of the tree is the starting symbol, and the edges represent the dependencies between terminal and non-terminals of the grammar as defined by the grammar's productions.

The approach presented here requires the construction of a *graph-isomorphism* between graphs of metamodel and abstract syntax tree. This ensures that we can go from the metamodel to the grammar in a way that preserves the information of the metamodel and back. Given graphs $G_i = (V_i, E_i, s_i, t_i)$, a *graph-morphism* is defined as (from [EEPT06]), $f : G_1 \rightarrow G_2$, where $f = (f_V, f_E)$ consists of two functions $f_V : V_1 \rightarrow V_2$ and $f_E : E_1 \rightarrow E_2$ that preserve the source and target functions (that is, $f_V \circ s_1 = t_2 \circ f_E$). f is called isomorphic if both functions f_V and f_E are bijections (both injective and surjective).

4.2 VCL Syntactic Isomorphisms

To show that that metamodel and grammar representations are equivalent, we need to show that there is an information-preserving isomorphism between the metamodels of common (Fig. 3.1),

SDs (Fig. 3.2) and ADs (Fig. 3.3) and the corresponding grammars of common (Fig. 3.4a), SDs (Fig. 3.4b) and ADs (Fig. 3.4c), respectively. These proofs are performed using the Z3 theorem prover [dMB08]¹; this involved encoding in Z3 the graphs of metamodel and grammar and all required theorems to prove. Z3 proves automatically all required theorems. The Z3 encoding of graphs and required theorems is given in appendix C.

4.2.1 Isomorphism Theorems and their Proofs

For each pair metamodel and grammar, several theorems need to be proved to demonstrate the existence of the information-preserving isomorphism as defined above. Let, G_{MM} and G_{Gr} be the graphs of metamodel and grammar respectively, such that: $G_{MM} = (V_{MM}, E_{MM}, s_{MM}, t_{MM})$ and $G_{Gr} = (V_{Gr}, E_{Gr}, s_{Gr}, t_{Gr})$. The two mapping functions of the isomorphism are: $f_V : V_{MM} \rightarrow V_{Gr}$ and $f_E : E_{MM} \rightarrow E_{Gr}$.

In the Z3 prover, the following theorems are proved. The source and target functions of both graphs must be total:

$$\begin{aligned} \forall emm : E_{MM} \bullet \exists vmm : V_{MM} \bullet s_{MM}(emm) = vmm \\ \forall emm : E_{MM} \bullet \exists vmm : V_{MM} \bullet t_{MM}(emm) = vmm \\ \forall egr : E_{Gr} \bullet \exists vgr : V_{Gr} \bullet s_{Gr}(egr) = vgr \\ \forall egr : E_{Gr} \bullet \exists vgr : V_{Gr} \bullet t_{Gr}(egr) = vgr \end{aligned}$$

The mapping functions must be total²:

$$\begin{aligned} \forall vmm : V_{MM} \bullet \exists vgr : V_{Gr} \bullet f_V(vmm) = vgr \\ \forall emm : E_{MM} \bullet \exists egr : E_{Gr} \bullet f_E(emm) = egr \end{aligned}$$

The mapping functions must be injective:

$$\begin{aligned} \forall vmm_1, vmm_2 : V_{MM} \bullet f_V(vmm_1) = f_V(vmm_2) \Rightarrow vmm_1 = vmm_2 \\ \forall emm_1, emm_2 : E_{MM} \bullet f_E(emm_1) = f_E(emm_2) \Rightarrow emm_1 = emm_2 \end{aligned}$$

The mapping functions must be surjective:

$$\begin{aligned} \forall vgr : V_{Gr} \bullet \exists vmm : V_{MM} \bullet f_V(vmm) = vgr \\ \forall egr : E_{Gr} \bullet \exists emm : E_{MM} \bullet f_E(emm) = egr \end{aligned}$$

All required Z3 encodings of graphs and theorems are given in appendix C.

Proofs for the common part

There is a direct isomorphism from the metamodel of common (Fig. 3.1) to the corresponding grammar (Fig. 3.4a). Further details of the Z3 proof are given in section C.1.1.

Proofs for the SD part

The SD metamodel of Fig. 3.2 requires a transformation into a another metamodel so that it is then possible to obtain a direct isomorphism. The transformed metamodel of SD that is isomorphic to the grammar of Fig. 3.4b is given in Fig. 4.1. Further details of the Z3 proof for the transformed metamodel are given in section C.2.

¹<http://research.microsoft.com/en-us/um/redmond/projects/z3/>

²In Z3, all functions definitions are total. To prove totality for the mapping functions in Z3, we resorted to a trick based on a special node called `Null`. The mapping functions are defined using Z3's `ite` (if-then-else) construct with the ultimate else being an assignment to the special `Null` node. A function is total provided there is one assignment to `Null`.

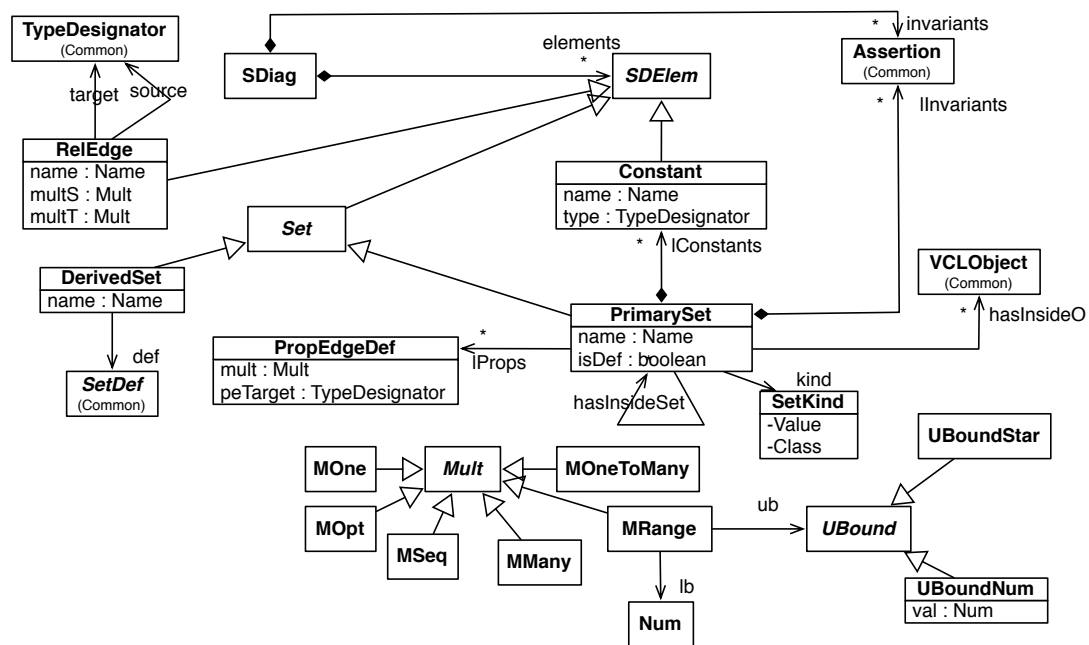


Figure 4.1: The transformed metamodel of SDs that is isomorphic to the grammar

Proofs for the AD part

There is a direct isomorphism from the metamodel of ADs (Fig. 3.3) to the corresponding grammar (Fig. 3.4c). Further details of the Z3 proof are given in section C.3.

Chapter 5

Type System

This chapter presents the type system of VCL structural and assertion diagrams. It starts by defining VCL's types and typing environments (section 5.1).

5.1 Types and Environments

A variable environment (VE) denotes a set of bindings, mapping identifiers to their types:

$$VE == Id \mapsto T$$

VCL's types (set T) are as follows:

$$T ::= \mathbf{Int} \mid \mathbf{Nat} \mid \mathbf{Null} \mid \mathbf{Pow} \ T \mid \mathbf{Seq} \ T \mid \mathbf{Opt} \ T \mid \mathbf{Top} \mid \mathbf{Obj} \mid \mathbf{Set} \ Id \mid \mathbf{Pair} \ (T, T) \\ \mid \mathbf{Assertion} \ [VE_v, VE_h]$$

Here, (a) **Int** represents the integers, (b) **Nat** the natural numbers; (c) **Null** represents erroneous results (implementation only); (d) **Pow** T represents a powerset of some set; (e) **Seq** T represents a sequence of some type; (f) **Opt** T represents an optional (either it exists or is empty); (g) **Top** is a maximal type (type of all well-formed terms); (h) **Obj** is the maximal type of all well-formed objects; (i) **Set** represents primary sets; (j) **Pair** represents a cartesian product of two types; (k) **Assertion** represents assertions (variable environments indicate assertion's variables, which are either visible, VE_v , or hidden, VE_h).

VCL's type rules use and manipulate environments (set E below), which are made of three components: (a) variable, (b) set and (c) subtyping. Variable environments give the type bindings of some scope. Set environments (SE) map identifiers to a triple made up of the set's kind (value or class), definitional status (DK) and local variable environment. Subtyping environments (set $SubE$) are the subtyping relations between types:

$$SK ::= \mathbf{value} \mid \mathbf{class} \\ DK ::= \mathbf{def} \mid \mathbf{notDef} \\ SE == Id \mapsto SK \times DK \times VE \\ SubE == T \leftrightarrow T \\ E == VE \times SE \times SubE$$

We introduce the following conventions:

Table 5.1 Judgements associated with the base rules of VCL's type system

$E \vdash T$	T is well-formed type in E
$E \vdash T_1 <: T_2$	T_1 is a subtype of T_2 in E
$E \vdash Id : T$	Id is well-formed identifier of type T in E
$E \vdash Id_s.Id_l : T$	Id_l is well-formed local identifier of set Id_s with type T in E

Table 5.2 Basic VCL typing rules

(Ty Id)	(Type)	(Ty LId)
$\frac{E.VE(Id) = T}{E \vdash Id : T}$	$\frac{T = \mathbf{Set} Id \Rightarrow Id \in \text{dom } E.SE}{E \vdash T}$	$\frac{E \vdash \mathbf{Set} Id_s \quad Id_l \in \text{dom}(E.SE(Id_s)).VE}{E \oplus (E.SE(Id_s)).VE \vdash Id_l : T}$

- \overline{X} and \widehat{X} denote, respectively, a sequence and a set of some set X .
- E_\emptyset is an empty environment. $E.VE$, $E.PE$, $E.SE$ and $E.SubE$ denote the different components of E .
- $Id : T$ and $Id \mapsto^{se} (SK, DK, Id, VE)$ are type (VE) and set (SE) bindings. $T_1 <: T_2$ says that T_1 is a subtype of T_2 .
- Disjoint environments are combined using E_1, E_2 ; similarly for other types of environments. Bindings are added to an environment using $E, Id : T$; similarly for other types of bindings. $E \oplus VE$ means that the environment E is overridden with the set of type bindings VE ; similarly for other types of bindings. These operators are defined precisely in appendix A.1.

5.2 Base Rules

The base type rules of VCL's type system manipulate environments and define subtype relations. The judgements are listed in table 5.1. The first judgement asserts that the type T is well-formed in the environment E . The second judgement asserts that the type T_1 is a subtype of T_2 in the environment E . The third judgement says that Id is a well-formed identifier with type T in E . The fourth judgement asserts the well-formedness of a set-property access; it says that property Id_l of set named Id_s has type T in E .

Table 5.2 lists basic rules concerning types. Rule **Ty Id** says that some identifier yields type T provided the variable binding is defined in the variable environment ($E.VE$). Rule **Type** describes the conditions for some type to be valid in some environment E : set types are valid provided their identifiers are defined in the set environment; all remaining types are valid. Rule **Ty LId** yields the type associated with some local identifier Id_l of some set Id_s ; the rule checks that the set type is defined and then retrieves the type of the local identifier from the set's variable environment.

Table 5.3 lists basic subtyping rules. Rule **Sub Ty** checks whether some type is a subtype of another; this amounts to check that both types are defined and that the subtyping tuple belongs to the environment's set of subtypes ($E.SubE$). Rules **Sub Refl** and **Sub Trans** says that the subtyping relation is both reflexive and transitive. Rule **Subsumption** is the subsumption rule that says that if some variable has type T_A , and if T_A is subtype of T_B then the variable also

Table 5.3 Basic VCL sub-typing rules

(Sub Ty)	(Sub Refl)	(Sub Trans)	(Subsumption)	(Sub Top)
$\frac{E \vdash T_1 \quad E \vdash T_2}{(T_1, T_2) \in E.SubE}$	$\frac{E \vdash T}{E \vdash T <: T}$	$\frac{E \vdash T_A <: T_B \quad E \vdash T_B <: T_C}{E \vdash T_A <: T_C}$	$\frac{E \vdash I : T_A}{E \vdash T_A <: T_B}$	$\frac{E \vdash T}{E \vdash T <: Top}$
(Sub Obj)	(Sub NatInt)	(Sub Pow)	(Sub Seq)	
$\frac{E \vdash \mathbf{Set} Id_s}{E \vdash \mathbf{Set} Id_s <: Obj}$	$\frac{}{E \vdash Nat <: Int}$	$\frac{}{E \vdash T_A <: T_B} \quad \frac{}{E \vdash \mathbf{Pow} T_A <: \mathbf{Pow} T_B}$	$\frac{}{E \vdash T_A <: T_B} \quad \frac{}{E \vdash \mathbf{Seq} T_A <: \mathbf{Seq} T_B}$	
(Sub Opt)	(Sub Opt PSet)	(Sub Pair)		
$\frac{}{E \vdash T_A <: T_B} \quad \frac{}{E \vdash \mathbf{Opt} T_A <: \mathbf{Opt} T_B}$	$\frac{}{E \vdash T_A <: T_B} \quad \frac{}{E \vdash \mathbf{Opt} T_A <: \mathbf{Pow} T_B}$	$\frac{}{E \vdash T_{A1} <: T_{A2} \quad E \vdash T_{B1} <: T_{B2}} \quad \frac{}{E \vdash \mathbf{Pair} (T_{A1}, T_{B1}) <: \mathbf{Pair} (T_{A2}, T_{B2})}$		

Table 5.4 Judgements of syntactic constructions common to VCL ADs and SDs

$E \vdash^{td} TD : T$	TD is well-formed type designator with type T in E
$E \vdash^{ta} A : Aid : T$	A is well-formed assertion with identifier Aid and type T in E
$E \vdash^{se} SE : T$	SE is well-formed set element with type T in E
$E \vdash^{sdef} SDef : T$	Set definition $SDef$ yields type T in E
$E \vdash^{id} IDef : T$	Inside definition $IDef$ yields type T in E
$E \vdash^{so} SOP(\overline{T}) : T$	Application of operator SOP to sequence of types \overline{T} yields type T
$E; T_s \vdash^{pe} PE : T'_s$	Property edge PE yields type T'_s
$E; T_s \vdash^{ppe} PEP : T'_s$	Predicate property edge PEP yields type T'_s
$E; T_s \vdash^{pem} PEM : T'_s$	Modifier property edge PEM yields type T'_s
$E \vdash^{te} TExp : T$	Target expression $TExp$ yields type T in E
$E \vdash^{ueo} UEOp(T_1) : T_2$	Application of unary edge operator $UEOp$ to type T_1 yields type T_2
$E \vdash^{eo} BEOP(T_1 T_2)$	Application of predicate edge operator $BEOP$ to types T_1, T_2 is well-typed
$E; \vdash^{mo} SOP(T_1 T_2) : T$	Application of set definition operator SOP to types T_1, T_2 yields type T

has type T_B . The remaining rules say how different types are subtyping related. Rule **Sub Top** says that any valid type is a subtype of **Top**. Rule **Sub Obj** says that any set type is a subtype of **Obj**. Rule **Sub NatInt** says that type of natural numbers is a subtype of the integers. Rules **SubPow**, **Sub Seq** and **Sub Opt** say, respectively, that two powerset, sequence or optional types are subtypes of each other provided their enclosed types (T_A and T_B) are also. Rule **Sub Opt PSet** says that optional types are a subtype of powerset types provided their enclosed types are also. Finally, rule **Sub Pair** says that two pair types are subtypes of each other if their corresponding components are also subtypes of each other.

5.3 Common Rules

The judgements for the syntactic constructions that are common to SDs and ADs (grammar or Fig. 3.4a) are given in table 5.4.

Type designator rules (Table 5.5) derive a type from a designator, yielding a primitive type (**Int** or **Nat**) or some type that is associated with an identifier (rule **TD Id**).

Table 5.6 presents rules for checking the well-formedness of assertions (**T Assertion**), VCL objects (**T SE Obj**) and pairs (**T SE Pair**). These rules merely extracts the types associated

Table 5.5 Type rules for type designators (TD non-terminal)

(TD Nat)	(TD Int)	(TD Id)
$\frac{}{E \vdash^{td} \mathbf{Nat} : Nat}$	$\frac{}{E \vdash^{td} \mathbf{Int} : Int}$	$\frac{E \vdash Id : T}{E \vdash^{td} Id : T}$

Table 5.6 Type rules for assertions and set elements

(T Assertion)	(T SE Obj)	(T SE Pair)
$\frac{E \vdash Id : T}{T = \mathbf{Assertion}[VE_v, VE_h]}$	$\frac{(IdS = Id \vee IdS = \mathbf{self})}{E \vdash IdS : T} \quad T <: \mathbf{Obj}$	$\frac{(IdS = Id \vee IdS = \mathbf{self})}{E \vdash IdS_1 : T_1 \quad E \vdash IdS_2 : T_2}$
$\frac{}{E \vdash^{ta} \mathbf{assertion} Id \therefore Id : T}$	$\frac{}{E \vdash^{se} \mathbf{object} IdS \therefore IdS : T}$	$\frac{}{E \vdash^{se} \mathbf{pair}(IdS_1, IdS_2) : \mathbf{Pair} (T_1, T_2)}$

with identifiers from the environment. The assertion rule assumes that the AD associated with the assertion being checked has already been type-checked and its information can, therefore, be retrieved from the environment. Pair rule builds a pair type from the types of its constituent identifiers.

A *set definition* (**SDef** nonterminal, Fig. 3.4a) is a syntactic construct to build sets. The type rules for set definitions (table 5.7) consider two cases, depending on whether the inside expression comprises one inside definition (rule **T SDef IDef**) or a sequence of set definitions (rule **T SDef SDef**). The rule essentially derive a sequence of types from inside definition (*IDef*) or sequence of set definitions (*SDef*) and then apply the rule for the set definition's operator (*SOp*) to retrieve the types yielded by the rules. An inside definition (**IDef** nonterminal, Fig. 3.4a) is a construction associated with set definitions. An inside definition can either be a constrained set or a set expression. The type rules for inside definitions (table 5.7) consider these two cases. The constrained set rule (**IDef CntSet**) derives a type from the given type designator (*TD*) and then checks the sequence of property edges in the context of this derived type (*T*); the rule says that the set of property edges must either be of only one kind: either predicate or modifier (disjunction). The type rules for set extensions (**IDef SE** and **IDef SE ***) process the sequence of set elements inductively; retrieving the greatest type of all the elements in the sequence, which must be subtypes of each other.

The rules for set definition operators (**SOp** non-terminal) apply to a sequence of types in the context of an environment and a set definition operator; they are given in table 5.8. The rules are as follows:

- Rule **SOp None** considers the case where there is no operator. The rule requires that the sequence of types is made of a single element, and yields the type given in the sequence.
- Rules for domain and range operators (**SOp Dom** and **SOp Ran**) require that there is a single type given in the sequence and that this type is a powerset of a pair (it is a binary relation). Rule **SOp Dom** returns a type formed as the powerset of the first type of the pair (the domain). Rule **SOp Ran** returns a type formed as the powerset of the second type (the range).
- The cross product rules (**SOp Cross** and **SOp * Cross**) consider two cases depending on whether the sequence is made of a pair of types or more than a pair. The pair rule takes

Table 5.7 Type rules for set definitions and associated inside definitions

$\frac{(T \text{ SDef } IDef) \quad E \vdash^{id} IDef : T_1 \quad E \vdash^{so} SOp(T_1) : T}{E \vdash^{sdef} \text{set} \bigcirc SOp \text{ hasIn } \{IDef\} : T}$		$\frac{(T \text{ SDef } \overline{SDef}) \quad E \vdash^{sdef} \overline{SDef}; : \overline{T_{sd}} \quad E \vdash^{so} SOp(\overline{T_{sd}}) : T_f}{E \vdash^{sdef} \text{set} \bigcirc SOp \text{ hasIn } \{\overline{SDef};\} : T_f}$	
(IDef CntSet)	(IDef SE)	(IDef $\overline{SE} *$)	
$\frac{E \vdash^{td} TD : T \quad E; T \vdash^{pe} \overline{PE} : T'}{(IsPEP(\overline{PE}) \vee IsPEM(\overline{PE}))}$		$\frac{E \vdash^{se} SE : T_1 \quad E \vdash^{id} \overline{SE} : T_2 \quad (T_r = T_1 \wedge T_2 <: T_1) \vee (T_r = T_2 \wedge T_1 <: T_2)}{E \vdash^{id} SE \overline{SE} : \mathbf{Pow} T_r}$	
$\frac{E \vdash^{id} \text{set } TD \{\overline{PE}\} : \mathbf{Pow} T'}{E \vdash^{id} SE : \mathbf{Pow} T}$			

a pair of powerset types and yields a powerset of a pair type. Rule **SOp * Cross** takes a powerset type and a sequence of types and returns a powerset of a pair type formed with the derived type.

- The intersection (**SOp Pair Intersection** and **SOp * Intersection**) and union rules (**SOp Pair Union** and **SOp * Union**) take a sequence of at least two powerset types and return a powerset of the greatest type in the sequence, according to the subtyping relation (function *getGType*, appendix A). All given types must be subtypes of each other. The set subtraction rule (**SOp Pair SetMinus**) does the same for a pair of powerset types.

Table 5.8 Type rules for set def operators

$(SOp \text{ None})$		$(SOp \text{ Dom})$	
$\frac{}{E \models^{\text{so}} \perp (T) : T}$		$\frac{}{E \models^{\text{so}} \leftarrow (\mathbf{Pow} \mathbf{Pair} (T_d, T_r)) : \mathbf{Pow} T_d}$	
$(SOp \text{ Ran})$		$(SOp \text{ RelComp})$	
$\frac{}{E \models^{\text{so}} \rightarrow (\mathbf{Pow} \mathbf{Pair} (T_d, T_r)) : \mathbf{Pow} T_r}$		$\frac{}{E \models^{\text{so}} \boxtimes (\mathbf{Pow} \mathbf{Pair} (T_1, T_2) \mathbf{Pow} \mathbf{Pair} (T_2, T_3)) : \mathbf{Pow} \mathbf{Pair} (T_1, T_3)}$	
$(SOp \text{ Cross})$		$(SOp \text{ Pair Intersection})$	
$\frac{}{E \models^{\text{so}} \times (\mathbf{Pow} T_1 \mathbf{Pow} T_2) : \mathbf{Pow} \mathbf{Pair} (T_1, T_2)}$		$\frac{T = \text{getGType}(E, T_1, T_2)}{E \models^{\text{so}} \cap (\mathbf{Pow} T_1 \mathbf{Pow} T_2) : \mathbf{Pow} T}$	
$(SOp * \text{ Cross})$		$(SOp * \text{ RelComp})$	
$\frac{}{E \models^{\text{so}} \times (\overline{T}) : \mathbf{Pow} T_2}$		$\frac{}{E \models^{\text{so}} \boxtimes (\overline{T}) : \mathbf{Pow} \mathbf{Pair} (T_2, T_3)}$	
$\frac{}{E \models^{\text{so}} \times (\mathbf{Pow} T_1 \overline{T}) : \mathbf{Pow} \mathbf{Pair} (T_1, T_2)}$		$\frac{}{E \models^{\text{so}} \boxtimes (\mathbf{Pow} \mathbf{Pair} (T_1, T_2) \overline{T}) : \mathbf{Pow} \mathbf{Pair} (T_1, T_3)}$	
$(SOp * \text{ Intersection})$		$(SOp \text{ Pair Union})$	
$\frac{}{E \models^{\text{so}} \cap (\overline{T}^1) : \mathbf{Pow} T_2}$		$\frac{T = \text{getGType}(E, T_1, T_2)}{E \models^{\text{so}} \cup (\mathbf{Pow} T_1 \mathbf{Pow} T_2) : \mathbf{Pow} T}$	
$\frac{}{E \models^{\text{so}} \cap (\mathbf{Pow} T_1 \overline{T}^1) : \mathbf{Pow} T}$		$\frac{T = \text{getGType}(E, T_1, T_2)}{E \models^{\text{so}} \setminus (\mathbf{Pow} T_1 \mathbf{Pow} T_2) : \mathbf{Pow} T}$	
$(SOp * \text{ Union})$		$(SOp \text{ Pair SetMinus})$	
$\frac{}{E \models^{\text{so}} \cup (\overline{T}^1) : \mathbf{Pow} T_2}$		$\frac{T = \text{getGType}(E, T_1, T_2)}{E \models^{\text{so}} \setminus (\mathbf{Pow} T_1 \mathbf{Pow} T_2) : \mathbf{Pow} T}$	
$\frac{}{E \models^{\text{so}} \cup (\mathbf{Pow} T_1 \overline{T}^1) : \mathbf{Pow} T}$			

Table 5.9 Type rules for property edges

$(PE \text{ PEP})$		$(PE \text{ PEM})$		$(PE *)$	
$\frac{E \models^{\text{te}} TExp : T_t}{E; (T_s, T_t) \models^{\text{pep}} PEP}$		$\frac{E \models^{\text{te}} TExp : T_t}{E; (T_s, T_t) \models^{\text{pem}} PEM} : T'_s$		$\frac{E; T_s \models^{\text{pe}} PE : T''_s}{E; T'_s \models^{\text{pe}} \overline{PE}^1 : T'_s}$	
$\frac{}{E; T_s \models^{\text{pe}} PEP TExp : T_s}$		$\frac{}{E; T_s \models^{\text{pe}} PEM TExp : T'_s}$		$\frac{}{E; T_s \models^{\text{pe}} PE \overline{PE}^1 : T'_s}$	
(PEP)		$(PEPS \epsilon)$		$(PEPS \text{ PrId})$	
$\frac{E; T_s \models^{\text{peps}} [Id] : T''_s}{E \models^{\text{ueo}} UEOp(T'_s) : T'_s}$		$\frac{}{E; T \models^{\text{peps}} \epsilon : T}$		$\frac{T = \mathbf{Set} Id_s}{E \models Id_s.Id_{pr} : T_p}$	
$\frac{}{E; (T_s, T_t) \models^{\text{pep}} UEOp [Id] \rightarrow [BEOp]}$		$\frac{}{E; T \models^{\text{peps}} \epsilon : T}$		$\frac{E \models^{\text{mo}} SOp(T_s T_t) : T}{E; (T_s, T_t) \models^{\text{pem}} [SOp] \Rightarrow : T}$	
$\frac{}{E \models^{\text{ueo}} BEOP(T'_s T_2)}$		$\frac{}{E; T \models^{\text{peps}} Id_{pr} : T_p}$		$\frac{}{E; (T_s, T_t) \models^{\text{pem}} [SOp] \Rightarrow : T}$	
$(UEOp \text{ No})$		$(UEOp \text{ Card})$		$(UEOp \text{ The})$	
$\frac{}{E \models^{\text{ueo}} \epsilon(T) : T}$		$\frac{}{E \models^{\text{ueo}} \# (\mathbf{Pow} T) : \mathbf{Int}}$		$\frac{}{E \models^{\text{ueo}} \odot (\mathbf{Opt} T) : T}$	
$\frac{}{E \models^{\text{ueo}} \# (\mathbf{Pow} T) : \mathbf{Int}}$		$\frac{}{E \models^{\text{ueo}} \odot (\mathbf{Opt} T) : T}$		$\frac{}{E \models^{\text{ueo}} \text{elems Seq } T : \mathbf{Pow} T}$	

Table 5.10 Type rules for binary predicate edge operators (BEOp)

$(BEOp \text{ EQNEQ})$		$(BEOp \text{ IN})$	
$\frac{(E \vdash T_1 <: T_2 \vee E \vdash T_2 <: T_1) \quad BEOp \in \{\neq, =\}}{E \models^{\text{eo}} BEOp(T_1 T_2)}$		$\frac{E \vdash T_1 <: T_2 \vee E \vdash T_2 <: T_1}{E \models^{\text{eo}} \in (T_1 \mathbf{Pow} T_2)}$	
$(BEOp \text{ INEQ})$		$(BEOp \text{ SUBSETEQ})$	
$\frac{E \vdash T_1 <: \text{Int} \quad E \vdash T_2 <: \text{Int} \quad BEOp \in \{<, \leq, >, \geq\}}{E \models^{\text{eo}} BEOp(T_1 T_2)}$		$\frac{E \vdash E \vdash T_1 <: T_2 \vee E \vdash T_2 <: T_1}{E \models^{\text{eo}} \subseteq (\mathbf{Pow} T_1 \mathbf{Pow} T_2)}$	

Table 5.11 Type rules for modifier edge operators (MOp)

$(MOp\ DRES)$	
$\frac{E \vdash T :< T_d}{E \vdash^{mo} \triangleleft (\mathbf{Pow\ Pair}\ (T_d, T_r), \mathbf{Pow}\ T) : \mathbf{Pow\ Pair}\ (T_d, T_r)}$	
$(MOp\ RRES)$	
$\frac{E \vdash T :< T_r}{E \vdash^{mo} \triangleright (\mathbf{Pow\ Pair}\ (T_d, T_r), \mathbf{Pow}\ T) : \mathbf{Pow\ Pair}\ (T_d, T_r)}$	
$(MOp\ DSUB)$	
$\frac{E \vdash T :< T_d}{E \vdash^{mo} \boxtimes (\mathbf{Pow\ Pair}\ (T_d, T_r), \mathbf{Pow}\ T) : \mathbf{Pow\ Pair}\ (T_d, T_r)}$	
$(MOp\ RSUB)$	
$\frac{E \vdash T :< T_r}{E \vdash^{mo} \boxtimes (\mathbf{Pow\ Pair}\ (T_d, T_r), \mathbf{Pow}\ T) : \mathbf{Pow\ Pair}\ (T_d, T_r)}$	
$(MOp\ RIMG)$	
$\frac{E \vdash T :< T_d}{E \vdash^{mo} \llbracket \rrbracket (\mathbf{Pow\ Pair}\ (T_d, T_r), \mathbf{Pow}\ T) : \mathbf{Pow}\ T}$	
$(MOp\ UNION)$	
$\frac{T = GetGType(E, T_1, T_2) \quad (T_1 <: T_2 \vee T_2 <: T_1)}{E \vdash^{mo} \cup (\mathbf{Pow}\ T_1, \mathbf{Pow}\ T_2) : \mathbf{Pow}\ T}$	
$(MOp\ INTERSEC)$	
$\frac{T = GetGType(E, T_1, T_2) \quad (T_1 <: T_2 \vee T_2 <: T_1)}{E \vdash^{mo} \cap (\mathbf{Pow}\ T_1, \mathbf{Pow}\ T_2) : \mathbf{Pow}\ T}$	
$(MOp\ SETMINUS)$	
$\frac{T = GetGType(E, T_1, T_2) \quad (T_1 <: T_2 \vee T_2 <: T_1)}{E \vdash^{mo} \setminus (\mathbf{Pow}\ T_1, \mathbf{Pow}\ T_2) : \mathbf{Pow}\ T}$	

Table 5.12 Type rules for set expressions

$(SExp\ TD)$	$(SExp\ Iden)$	$(SExp\ SDef)$
$\frac{E \vdash^{td} TD : T}{E \vdash^{te} \mathbf{set}\ TD : \mathbf{Pow}\ T}$	$\frac{E \vdash^{te} \mathbf{set}\ \mathbf{iden} : \mathbf{Pow\ Pair}\ (\mathbf{Top}, \mathbf{Top})}{E \vdash^{te} \mathbf{set}\ \mathbf{iden} : \mathbf{Pow}\ T}$	$\frac{E \vdash^{sdef} SDef : T}{E \vdash^{te} SDef : T}$
$(SExp\ Empty)$	$(SExp\ Card)$	$(SExp\ elems)$
$\frac{E \vdash^{te} \mathbf{set}\ \mathbf{shaded} : \mathbf{Pow}\ \mathbf{Top}}{E \vdash^{te} \mathbf{set}\ \mathbf{shaded} : \mathbf{Pow}\ \mathbf{Top}}$	$\frac{E \vdash^{te} SExp : \mathbf{Pow}\ T}{E \vdash^{te} \# SExp : \mathbf{Int}}$	$\frac{E \vdash^{te} SExp : \mathbf{Seq}\ T}{E \vdash^{te} \mathbf{elems}\ SExp : \mathbf{Pow}\ T}$

Table 5.13 Type rules for free expressions

(FExp ID)	(FExp Dot)	(FExp Num)	(FExp Uminus)
$(IdS = Id \vee IdS = \mathbf{self})$	$(IdS_o = Id \vee IdS_o = \mathbf{self})$		
$\frac{E \vdash IdS : T}{E \vdash^{te} IdS : T}$	$\frac{E \vdash IdS_o : \mathbf{Set}\ Id_s \quad E \vdash Id_s.Id_{pr} : T}{E \vdash^{te} IdS_o.Id_{pr} : T}$	$\frac{E \vdash^{te} FE : Int}{E \vdash^{te} Num : Nat}$	$\frac{E \vdash^{te} FE : Int}{E \vdash^{te} -FE : Int}$
(FExp FEOP)			
$\frac{E \vdash^{te} FE_1 : Int \quad E \vdash^{te} FE_2 : Int}{E \vdash^{te} FE_1\ FEOP\ FE_2 : Int}$			

Table 5.14 Judgements for type system of VCL Structural Diagrams

$E; \overline{AD} \vdash^{sd} SD \therefore E'$	SD yields environment E'
$E; \overline{AD} \vdash^{sde} \overline{SDE} \therefore E'$	Sequence of SD elements \overline{SDE} yields environment E'
$E; \overline{AD}; Id_{s\perp} \vdash^{as} \overline{A} \therefore VE$	Sequence of assertions \overline{A} yields variable environment VE
$E \vdash^{cn} \overline{C} \therefore VE$	Sequence of constants \overline{C} yields variable environment VE
$E; \overline{AD}; T \vdash^{pset} PSet \therefore E'$	Primary Set $PSet$ yields environment E'
$E \vdash^{ped} \overline{PED} \therefore VE$	Sequence of edge definitions \overline{PED} yields variable environment VE
$E; M \vdash^{mtd} TD : T$	Designator TD with multiplicity M yields type T
$E; \overline{AD}; T \vdash^{hi} HI \therefore E'$	HI (<i>HasIn</i>) yields environment E'
$E; T \vdash^{io} \overline{O} \therefore VE$	Sequence of inside objects \overline{O} yields variable environment VE
$E; T \vdash^{is} \overline{PSet} \therefore E'$	Sequence of inside primary sets \overline{PSet} yields environment E'
$E; \overline{AD}; Sid_{\perp} \vdash^{aok} A \therefore Aid : T$	A has a well-formed assertion diagram with identifier Aid type T in E
$E; \overline{AD}; Id_{s\perp} \vdash^{adok} \widehat{\overline{AD}} \therefore VE$	$\widehat{\overline{AD}}$ is set of ADs yielding variable environment VE

5.4 Rules for Structural Diagrams

Table 5.14 presents the judgements for structural diagrams (SDs). The first judgement says that a SD is well-formed in the environment E with environment E' . The remaining judgements assert well-formedness for the different components of a SD; namely, sequences of structural diagram element (judgement labelled \vdash^{sde}), sequences of assertions denoting invariants (\vdash^{as}), sequences of constants (\vdash^{cn}), primary sets (\vdash^{pset}), sequence of property edge definitions (\vdash^{ped}), designators with a multiplicity constraint (\vdash^{mtd}), has inside declarations of primary sets (\vdash^{hi}), sequence of inside objects (\vdash^{io}), sequences of inside primary sets (\vdash^{is}), assertion whose AD has not been checked (\vdash^{aok}) and set of ADs (\vdash^{adok}).

Table 5.15 Type rules for structural diagrams and sequences of diagram elements

(Ok SD)	$(\overline{SDE} \ *)$	$(\overline{SDE} \ \epsilon)$
$E; \overline{AD} \vdash^{sde} \overline{SDE} \therefore E'$		
$Acyclic E'.SE$	$E; \overline{AD} \vdash^{sde} SDE \therefore E'$	
$E, E'; \overline{AD}; \perp \vdash^{as} \overline{A} \therefore VE$	$E, E'; \overline{AD} \vdash^{sde} \overline{SDE} \therefore E''$	
$\frac{E; \overline{AD} \vdash^{sd} \overline{SDE} \overline{A} \therefore E, E', VE}{E; \overline{AD} \vdash^{sde} \overline{SDE} \overline{SDE} \therefore E, E', E''}$	$\frac{E; \overline{AD} \vdash^{sde} SDE \overline{SDE} \therefore E, E', E''}{E; \overline{AD} \vdash^{sde} \epsilon \therefore E_{\emptyset}}$	

Table 5.16 Type rules for constants, relation edges and sets

$\frac{(SDE\ Const) \quad E \vdash^{cn} C \therefore VE_c}{E; \overline{AD} \vdash^{sde} C \therefore E_\emptyset, VE_c}$		$\frac{(SDE\ RelEdge) \quad E \vdash^{td} TD_1 : T_1 \quad E \vdash^{td} TD_2 : T_2 \quad M_1 \neq \mathbf{seq} \quad M_2 \neq \mathbf{seq}}{E; \overline{AD} \vdash^{sde} \mathbf{relEdge}\ Id_{RE}(M_1\ TD_1, M_2\ TD_2) \therefore E_\emptyset, Id_{RE} : \mathbf{Pow\ Pair}\ (T_1, T_2)}$	
$(Const) \quad \frac{E \vdash^{td} TD : T}{E \vdash^{cn} \mathbf{const}\ Id_{Cn} : TD \therefore \{Id_{Cn} : T\}} \quad \frac{E \vdash^{cn} \epsilon \therefore \{\}}{E \vdash^{cn} \epsilon \therefore \{\}} \quad \frac{E \vdash^{cn} C \therefore VE_1 \quad E \vdash^{cn} \overline{C} \therefore VE_2 \quad \text{dom } VE_1 \cap \text{dom } VE_2 = \{\}}{E \vdash^{cn} C\ \overline{C} \therefore VE_1, VE_2}$			
$(SDE\ PSet) \quad \frac{E; \overline{AD}; \mathbf{Obj} \vdash^{pset} PSet \therefore E_b}{E; \overline{AD} \vdash^{sde} PSet \therefore E_b}$	$(SDE\ Derived) \quad \frac{E \vdash^{sdef} SDef : T}{E; \overline{AD} \vdash^{sde} Id_s \leftrightarrow SDef \therefore E_\emptyset, Id_s : T}$		

Table 5.17 Type rules for primary sets

$(Primary\ Set) \quad \frac{Id_s \notin E.VE \quad E \vdash^{cn} \overline{C} \therefore VE_c \quad E \vdash^{ped} \overline{PED} \therefore VE_{pe} \quad E; \overline{AD}; Id_s \vdash^{as} \overline{A} \therefore VE_a \quad VE_i = getVE(E, T) \quad T_s = \mathbf{Set}\ Id_s \quad DK = getDK([\bigcirc]) \quad SI = (SK, DK, (VE_c, VE_{pe}, VE_a, VE_i))}{E, Id_s : \mathbf{Pow}\ T_s, Id_s \mapsto SI; \overline{AD}; T_s \vdash^{hi} [\mathbf{hasIn}\ \{\overline{(O\ \ PSet)}\}] \therefore (E_{hi})}$	
$E; \overline{AD}; T \vdash^{pset} \mathbf{set}\ Id_s\ SK\ [\bigcirc]\ \{\overline{C\ PED\ A}\} [\mathbf{hasIn}\ \{\overline{(O\ \ PSet)}\}] \therefore (E_\emptyset, Id_s : \mathbf{Pow}\ T_s, Id_s \mapsto SI, T_s <: T, E_{hi})$	

Table 5.18 Type rules for property edge definitions

$(\overline{PED}\ \epsilon) \quad \frac{E \vdash^{ped} \epsilon \therefore VE_\emptyset}{E \vdash^{ped} \epsilon \therefore VE_\emptyset}$	$(\overline{PED}\ *) \quad \frac{E; M \vdash^{mtd} TD : T \quad E \vdash^{ped} \overline{PED} \therefore VE_2}{E \vdash^{ped} M\ Id_{Pe} \rightarrow TD\ \overline{PED} \therefore \{Id_{Pe} : T\}, VE_2}$
$(MTD\ One) \quad \frac{E \vdash^{td} TD \therefore T}{E; \mathbf{one} \vdash^{mtd} TD : T}$	$(MTD\ Pow) \quad \frac{E \vdash^{td} TD \therefore T \quad M = \mathbf{some} \vee M = \mathbf{many} \vee M = Num \dots (Num *)}{E; M \vdash^{mtd} TD : \mathbf{Pow}\ T}$
$(MTD\ Opt) \quad \frac{E \vdash^{td} TD \therefore T}{E; \mathbf{opt} \vdash^{mtd} TD : \mathbf{Opt}\ T}$	$(MTD\ Seq) \quad \frac{E \vdash^{td} TD \therefore T}{E; \mathbf{seq} \vdash^{mtd} TD : \mathbf{Seq}\ T}$

Table 5.19 Type rules for sequences of invariants

$(\overline{A}\ \epsilon) \quad \frac{E; \overline{AD}; Id_{s\perp} \vdash^{as} \epsilon \therefore \{\}}{E; \overline{AD}; Id_{s\perp} \vdash^{as} \epsilon \therefore \{\}}$	$(\overline{A}\ *) \quad \frac{E; \overline{AD}; Id_{s\perp} \vdash^{as} A \therefore VE_1 \quad E; \overline{AD}; Id_{s\perp} \vdash^{as} \overline{A} \therefore VE_2}{E; \overline{AD}; Id_{s\perp} \vdash^{as} A\ \overline{A} \therefore VE_1, VE_2}$
--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Table 5.20 Type rules for has inside declarations

$(HasInside \epsilon)$	$(HasInside *)$	$(HasInObjs \epsilon)$
$\frac{}{E; T \vdash^{hi} \epsilon \therefore E_{\emptyset}}$	$\frac{E; T \vdash^{io} \overline{O} \therefore VE \quad E; T \vdash^{is} \overline{PSet} \therefore E'}{E; T \vdash^{hi} \mathbf{hasIn} \{ \overline{O} \overline{PSet} \} \therefore E', VE}$	$\frac{}{E; T \vdash^{io} \epsilon \therefore \{ \}}$
$(HasInObjs *)$	$(HasInSet \epsilon)$	$(HasInSets *)$
$\frac{E; T \vdash^{io} \overline{O} \therefore VE}{E; T \vdash^{io} \mathbf{object} Id_o \overline{O} \therefore VE, \{ Id_o : T \}}$	$\frac{}{E; T \vdash^{is} \epsilon \therefore E_{\emptyset}}$	$\frac{E; T \vdash^{set} PSet \therefore E' \quad E, E'; T \vdash^{is} \overline{PSet} \therefore E''}{E; T \vdash^{is} PSet \overline{PSet} \therefore E', E''}$

Table 5.21 Type rules for checking assertions

$(AssertionOk)$	$(AD Ok)$
$\frac{Id_A \notin \text{dom } E.VE \quad AD = \text{findAD}(\overline{AD}, Id_A, Id_{s\perp}) \quad E; \overline{AD}; Id_{s\perp} \vdash^{adok} AD \therefore VE}{E; \overline{AD}; Id_{s\perp} \vdash^{adok} \mathbf{assertion} Id_A \therefore VE}$	$\frac{\widehat{AD} = \text{getDepsOfAD}(AD, \overline{AD}, Id_{s\perp}) \quad E; \overline{AD}; Id_{s\perp} \vdash^{adok} \widehat{AD} \therefore VE \quad E, VE; \overline{AD} \vdash^{ad} AD \therefore Id_A : T}{E; \overline{AD}; Id_{s\perp} \vdash^{adok} AD \therefore VE, Id_A : T}$
$(ADs Ok \in Local)$	$(ADs Ok \in Global)$
$\frac{Id_s \neq \perp \quad E \vdash Id_s : \mathbf{Pow} T \quad VE = \{ \mathbf{self} : T \}}{E; \overline{AD}; Id_{s\perp} \vdash^{adok} \{ \} \therefore VE}$	$\frac{Id_s = \perp}{E; \overline{AD}; Id_{s\perp} \vdash^{adok} \{ \} \therefore \{ \}}$
$(ADs Ok *)$	
$\frac{E; \overline{AD}; Id_{s\perp} \vdash^{adok} AD \therefore VE \quad E; \overline{AD}; Id_{s\perp} \vdash^{ads} \widehat{AD} \therefore VE'}{E; \overline{AD}; Id_{s\perp} \vdash^{adok} \{ AD \} \cup \widehat{AD} \therefore VE, VE'}$	

Table 5.21 presents the rules for checking ADs associated with some assertion. These rules are used when the AD type information is to be loaded into the environment. The rules are as follows:

- Rule **Assertion Ok** derives the name of the assertion diagram through function **getFAId**, which considers the special case of assertions associated with constants, and then looks for the AD using function **findAD** (both functions defined in appendix A, section A.3.5). The retrieved AD is then checked (rule associated with judgement \vdash^{adok}) to yield variable environment VE .
- Rule **AD Ok** processes a single AD. It retrieves all the ADs that are included in the given AD through function **getDepsOfAD** (appendix A, section A.3.4) to yield set \widehat{AD} and then checks them using the rules associated with judgement \vdash^{adok} to derive variable environment VE . The current AD is also checked using the rule for assertion diagrams to yield a variable binding. The rule yields a variable environment formed by adding the retrieved variable binding to the variable environment VE .
- Rules $AD Ok \in$ and $AD Ok *$ process a set of ADs inductively. Rule $AD Ok \in$ considers the

case where the set is empty, yielding an empty set of variable bindings. Rule $AD\ Ok^*$ considers the case where the set has at least one element; it builds a variable environment by joining the variable environment derived from the current single AD and the variable environment derived from the remaining set of ADs.

Table 5.22 Judgements for typing of assertion diagrams

$E \vdash^{ad} AD \therefore I : T$	AD yields binding $I : T$ in E
$E \vdash^{vd} \overline{VD} \therefore (VE_v, VE_h)$	Variable declarations block VD yields binding sets (VE_v, VE_h)
$E \vdash^d \overline{D} \therefore (VE_v, VE_h)$	Sequence of declarations \overline{D} yields binding sets (VE_v, VE_h)
$E \vdash^{df} DF \therefore (VE_v, VE_h)$	Declarations formula atom DF yields binding sets (VE_v, VE_h)
$E \vdash^f \overline{F}$	Sequence of formulas \overline{F} is well-formed in E
$E \vdash^{afs} AFS : T$	Arrows formula source AFS yields type T

Table 5.23 Type rules for assertion diagrams

$(AD\ GBL)$	
$E \vdash^d \overline{D} \therefore (VE_v; VE_h)$	$E \oplus (VE_v, VE_h) \vdash^f \overline{F}$
$E \vdash^{ad} \mathbf{AD}\ Id_A\ \mathbf{decls}\ \{\overline{D}\}\ \mathbf{pred}\ \{\overline{F}\} \therefore Id_A : \mathbf{Assertion}[VE_v, VE_h]$	
$(AD\ LOCAL)$	
$E \vdash Id_s : \mathbf{Pow\ Set}\ Id_s$	
$E.SE(Id_s) = (SK, DK, VE_s)$	$E \oplus VE_s \vdash^d \overline{D} \therefore (VE_v; VE_h)$
$E \oplus (VE_s, VE_v, VE_h) \vdash^f \overline{F}$	
$E \vdash^{ad} \mathbf{AD}\ Id_A : Id_s\ \mathbf{decls}\ \{\overline{D}\}\ \mathbf{pred}\ \{\overline{F}\} \therefore Id_A : \mathbf{Assertion}[VE_v, VE_h]$	

5.5 Rules for Assertion Diagrams

The judgements for ADs are listed in Table 5.22. In the judgements's contexts, E is an environment; the AD rules assume that all relevant ADs have been checked and its information can be found in the environment. The judgements are as follows. The first judgement (\vdash^{ad}) asserts the well-formedness of some AD, yielding a binding made up of the AD's identifier and type. The remaining judgements concern either the declarations or predicate compartment of ADs. The declarations judgements include: judgement \vdash^d , which says that a sequence of declarations (\overline{D}) is well-formed and \vdash^{df} , which says that a particular declaration formula (DF) is well-formed. The predicate compartment includes judgements for formulas (\vdash^f) and arrows formula source (\vdash^{afs}).

The typing rules for ADs (table 5.23) consider two cases, corresponding to global (**AD GBL**) and local ADs (**AD LOCAL**). The rules are similar: the typing of declarations is followed by the typing of the predicate. The local rule requires the local variable environment, which it retrieves from the set environment component of the environment ($E.SE$). The processing of the declaration yields two variable environments: the visible (VE_v) and the hidden (VE_h) variables. The visible variables are visible in the assertions predicate and to the outside world; the hidden variables are only visible within the assertion.

The type rules for the declarations (table 5.24) build the visible and hidden variable environments. They are follows:

- Rules $\overline{D} \epsilon$ and $\overline{D} *$ handle a sequence of declaration inductively. Rule $\overline{D} \epsilon$ yields the empty variable environments ($\{\}$) for both visible and hidden: there are no declarations to process. Rule $\overline{D} *$ retrieves the variable environments from the current declaration (VE_v, VE_h) and from the remaining declarations (VE_{vs}, VE_{hs}); the variables environments to be yielded by the rule are then merged (operator \bowtie), which requires that identifiers in common in the variable environments being combined must be bound to the same type; furthermore, all variables from the visible list (VE_{vf}) are removed in the hidden list (operator \boxtimes).
- Rules **D Obj** and **D Set** consider the cases where there is a declaration of a scalar (object)

Table 5.24 Type rules for declarations

$(\overline{D} \epsilon)$	$(\overline{D} *)$
$\frac{E \vdash^d \epsilon \therefore (\{\}; \{\})}{E \vdash^d \epsilon \therefore (\{\}; \{\})}$	$\frac{E \vdash^d D \therefore (VE_v; VE_h) \quad E, VE_v, VE_h \vdash^d \overline{D} \therefore (VE_{vs}; VE_{hs})}{VE_{vf} = VE_v \bowtie VE_{vs} \quad VE_{hf} = (VE_h \bowtie VE_{hs}) \boxtimes VE_{vf}}$
$(VD \text{ Obj})$	$(VD \text{ Set})$
$\frac{E \vdash^{td} TD : T \quad (OQ = \mathbf{opt} \wedge T_f = \mathbf{Opt} T \vee OQ = \epsilon \wedge T_f = T) \quad VE = \{\overline{Id_O}, : T_f\} \quad (HQ = \epsilon \wedge VE_v = VE \wedge VE_h = \{\}) \vee HQ = \mathbf{hidden} \wedge VE_v = \{\} \wedge VE_h = VE}{E \vdash^{vd} HQ \mathbf{object} OQ \overline{Id_O} : TD \therefore (VE_v, VE_h)}$	$\frac{E \vdash^{td} TD : T \quad VE = \{\overline{Id_s}, : \mathbf{Pow} T\} \quad (HQ = \epsilon \wedge VE_v = VE \wedge VE_h = \{\}) \vee HQ = \mathbf{hidden} \wedge VE_v = \{\} \wedge VE_h = VE}{E \vdash^{vd} HQ \mathbf{set} \overline{Id_s} : TD \therefore (VE_v, VE_h)}$
$(VD \text{ Seq})$	$(VD *)$
$\frac{E \vdash^{td} TD : T \quad VE = \{\overline{Id_s}, : \mathbf{Seq} T\} \quad (HQ = \epsilon \wedge VE_v = VE \wedge VE_h = \{\}) \vee HQ = \mathbf{hidden} \wedge VE_v = \{\} \wedge VE_h = VE}{E \vdash^{vd} HQ \mathbf{seq} \overline{Id_s} : TD \therefore (VE_v, VE_h)}$	$\frac{E \vdash^{vd} VD \therefore (VE_{v1}, VE_{h1}) \quad E \vdash^{vd} \overline{VD} \therefore (VE_{v2}, VE_{h2}) \quad VE_{v1} \cap VE_{v2} \cap VE_{h1} \cap VE_{h2} = \{\}}{E \vdash^{vd} VD \overline{VD} : TD \therefore (VE_{v1} \cup VE_{v2}, VE_{h1} \cup VE_{h2})}$
$(D \text{ VD})$	$(D \text{ DF})$
$\frac{E \vdash^{vd} VD \therefore (VE_v, VE_h)}{E \vdash^d VD \therefore (VE_v, VE_h)}$	$\frac{E \vdash^{df} DF \therefore (VE_v, VE_h)}{E \vdash^d DF \therefore (VE_v, VE_h)}$

or set. Both rules retrieves a type from the declaration's type designator (TD) and then yield a visible binding made of the variable's identifier and appropriate type. Rule **D Obj** considers whether there is an optional qualifier ; type to yield is optional if there is a qualifier (**Opt** T) or the type derived from the type designator otherwise (T). Rule **D Set** also considers whether there is a sequence qualifier; type to yield is sequence of there is a qualifier (**Seq** T) or a powerset otherwise (**Pow** T).

- Rule **D DF** considers the case where the declaration comprises a declarations formula. In this case, the type rule for declaration formulas is called.

Table 5.25 presents the type rules for declaration formulas. The rules are as follows:

- Rules **DFA Assertion**, **DFA OCall** and **DFA C1Call** deal with declaration formula atoms (**DFA** non-terminal, Fig. 3.4c). Rule **DFA Assertion** considers the case where the construction refers to a normal assertion defined in the same scope (either local or global); rule **DFA OCall** considers the case where there is a local assertion being called on some object; and rule **DFA C1Call** considers the case where a class assertion is called.
- Rules **DFA Assertion**, **DFA OCall** and **DFA C1Call** assume that the AD associated with the assertion being checked has already been type-checked: the assertion's type can be retrieved from the environment. These rules retrieve the appropriate assertion type from the environment to obtain the assertion's visible and hidden bindings (VE_v and VE_h). From the assertion's visible bindings (VE_v), the rule then builds the visible and hidden bindings for the declaration using function *conVEs*, which takes into account the presence of symbol \uparrow , and from these constructed bindings the rule makes the required substitutions

Table 5.25 Type rules for declaration formulas

<p>(DFA Assertion)</p> $\frac{E \vdash^{la} A \therefore Id_A : \mathbf{Assertion}[VE_v; VE_h] \quad (VE_{cv}, VE_{ch}) = consVES(VE_v, [\uparrow]) \quad (VE_{fv}, VE_{fh}) = doSubs(VE_{cv}, VE_{ch}, [\overline{R}_,])}{E \vdash^{df} [\uparrow]A[\overline{R}_,] \therefore (VE_{fv}; VE_{fh})}$	<p>(DFA OCall)</p> $\frac{E \vdash Id_O : T_s \quad Id_s = getIdFrTy(T_s) \quad E \vdash Id_s.Id_A : \mathbf{Assertion}[VE_v, VE_h] \quad (VE_{cv}, VE_{ch}) = consVES(VE_v, [\uparrow]) \quad (VE_{fv}, VE_{fh}) = doSubs(VE_{cv}, VE_{ch}, [\overline{R}_,])}{E \vdash^{df} [\uparrow]\mathbf{assertion} Id_O.Id_A[\overline{R}_,] \therefore (VE_{fv}; VE_{fh})}$
<p>(DFA ClCall)</p> $\frac{E \vdash Id_s.Id_A : \mathbf{Assertion}[VE_v, VE_h] \quad (VE_{cv}, VE_{ch}) = consVES(VE_v, [\uparrow]) \quad (VE_{fv}, VE_{fh}) = doSubs(VE_{cv}, VE_{ch}, [\overline{R}_,])}{E \vdash^{df} [\uparrow]\mathbf{assertion} Id_s \rightarrow Id_A[\overline{R}_,] \therefore (VE_{fv}; VE_{fh})}$	
<p>(DF Neg)</p> $\frac{E \vdash^{df} DF \therefore (VE_v; VE_h)}{E \vdash^{df} \neg [DF] \therefore (VE_v; VE_h)}$	<p>(DF Bin)</p> $\frac{E \vdash^{df} DF_1 \therefore (VE_{v1}; VE_{h1}) \quad E \vdash^{df} DF_2 \therefore (VE_{v2}; VE_{h2}) \quad FOp \in \{\Rightarrow, equiv\}}{E \vdash^{df} FOp[DF_1 DF_2] \therefore (VE_{v1} \bowtie VE_{v2}; VE_{h1} \bowtie VE_{h2})}$
<p>(DF NArY 2*)</p> $\frac{E \vdash^{df} \overline{DF} \therefore (VE_v; VE_h) \quad FOp \in \{\vee, \wedge, \boxtimes\} \quad \# \overline{DF} \geq 2}{E \vdash^{df} FOp[\overline{DF}] \therefore (VE_v; VE_h)}$	
<p>(DF NArY *)</p> $\frac{E \vdash^{df} DF \therefore (VE_{v1}; VE_{h1}) \quad E \vdash^{df} \overline{DF} \therefore (VE_{v2}; VE_{h2})}{E \vdash^{df} DF \overline{DF} \therefore (VE_{v1} \bowtie VE_{v2}; VE_{h1} \bowtie VE_{h2})}$	
<p>(DF NArY ϵ)</p> $\frac{E \vdash^{df} \epsilon \therefore (\{\}; \{\})}{E \vdash^{df} DF \overline{DF} \therefore (VE_{v1} \bowtie VE_{v2}; VE_{h1} \bowtie VE_{h2})}$	

according to what is defined in the sequence of renamings $(\overline{R}_,)$ using function *applySubs*. All it varies in the rules is the way the assertion type is obtained; rule **DFA Assertion** obtains the assertion type directly from the environment; rule **DFA OCall** obtains the assertion type from the object's set; and rule **DFA ClCall** obtains the assertion type from the given set identifier.

- Rule **DF Neg** obtains the visible and hidden variables of a negated declarations formula from the enclosed declarations formula.
- Rule **DF Bin** handles a binary declarations formula combined using a binary operator. The rules obtains the visible and hidden bindings from the two declarations formulas being combined and then merges them using the operator *mergeves*.

Table 5.26 Type rules for Formulas (F)

$(\overline{F} \epsilon)$	$(\overline{F} *)$	$(F \text{ Not})$	$(F \text{ Bin})$	$(F \text{ NAry})$	$(F \text{ AF})$
$\frac{}{E \vdash^f \epsilon}$	$\frac{E \vdash^f F}{E \vdash^f \overline{F}}$	$\frac{E \vdash^f F}{E \vdash^f \neg [F]}$	$\frac{E \vdash^f F_1 \quad E \vdash^f F_2 \quad FOp \in \{\Rightarrow, \Leftrightarrow\}}{E \vdash^f FOp[F_1 F_2]}$	$\frac{E \vdash^f \overline{F} \quad \# \overline{F} \geq 2 \quad FOp \in \{\wedge, \vee\}}{E \vdash^f FOp[\overline{F}]}$	$\frac{E \vdash^{afs} AFS : T \quad E; T \vdash^{eps} \overline{PEP}^1}{E \vdash^f AFS \{\overline{PEP}^1\}}$
$(F \text{ SF Shaded})$	$(F \text{ SF Id})$	$(F \text{ SF Inside})$			$(AFS \text{ SE})$
$\frac{E \vdash^{sdef} SDef : T}{E \vdash^f \text{shaded } SDef}$	$\frac{E \vdash Id_s : T_1 \quad E \vdash^{sdef} SDef : T_2 \quad (E \vdash T_2 <: T_1 \vee E \vdash T_2 <: T_1)}{E \vdash^f [\text{shaded}] Id_s SDef}$	$\frac{E \vdash^{td} TD : T_1 \quad E \vdash^{te} SExp : T_2 \quad (E \vdash T_1 <: T_2 \vee E \vdash T_2 <: T_1)}{E \vdash^f \text{set } TD \text{ hasIn } \{SExp\}}$			$\frac{E \vdash^{td} TD : \mathbf{Pow} T_2}{E \vdash^f \text{set shaded } TD}$
(QF)	(QD)	$(QD *)$			
$\frac{E \vdash^{gd} \overline{QD}, \therefore VE \quad E \oplus VE \vdash^f \overline{F}}{E \vdash^f \overline{QD}, \bullet \overline{F};}$	$\frac{E \vdash^{vd} \overline{VD} \therefore (VE_v, VE_h) \quad VE_h = \{\}}{E \vdash^{gd} Q \overline{VD}, \therefore VE_v}$	$\frac{E \vdash^{vd} \overline{VD} \therefore (VE_v, VE_h) \quad VE_h = \{\} \quad E \vdash^{gd} \overline{QD}, \therefore VE}{E \vdash^{gd} Q \overline{VD}, \overline{QD}, \therefore VE_v \oplus VE}$			

Table 5.27 Type rules for Arrows Formula Source (production *AFS*)

$(AFS \text{ SE})$	$(AFS \text{ SetId})$	$(AFS \text{ SDef})$	$(AFSB \text{ Un Card})$
$\frac{E \vdash^{se} SE : T}{E \vdash^{afs} SE : T}$	$\frac{E \vdash Id_s : T}{E \vdash^{afs} \text{set } Id_s : T}$	$\frac{E \vdash^{sdef} SDef : T}{E \vdash^{afs} SDef : T}$	$\frac{E \vdash^{afs} AFS : \mathbf{Pow} T}{E \vdash^{afs} \# AFS : \mathbf{Int}}$
$(AFS \text{ Un Dom})$	$(AFS \text{ Un Ran})$	$(AFSB \text{ Un The})$	
$\frac{E \vdash^{afs} AFS : \mathbf{Pow} \mathbf{Pair} (T_1, T_2)}{E \vdash^{afs} \leftarrow AFS : \mathbf{Pow} T_1}$	$\frac{E \vdash^{afs} AFS : \mathbf{Pow} \mathbf{Pair} (T_1, T_2)}{E \vdash^{afs} \rightarrow AFS : \mathbf{Pow} T_2}$	$\frac{E \vdash^{afs} AFS : \mathbf{Opt} T}{E \vdash^{afs} \odot AFS : T}$	

References

- [AGK11] Nuno Amálio, Christian Glodt, and Pierre Kelsen. Building VCL models and automatically generating Z specifications from them. In *FM 2011*, number 6664 in LNCS, pages 149–153. Springer, 2011.
- [AK10] Nuno Amálio and Pierre Kelsen. Modular design by contract visually and formally using VCL. In *VL/HCC 2010*, 2010.
- [AKMG10] Nuno Amálio, Pierre Kelsen, Qin Ma, and Christian Glodt. Using VCL as an aspect-oriented approach to requirements modelling. *TAOSD*, VII:151–199, 2010.
- [Amá07] Nuno Amálio. *Generative frameworks for rigorous model-driven development*. PhD thesis, Dept. Computer Science, Univ. of York, 2007.
- [APS05] Nuno Amálio, Fiona Polack, and Susan Stepney. An object-oriented structuring for Z based on views. In *ZB 2005*, volume 3455 of *LNCS*, pages 262–278. Springer, 2005.
- [dMB08] Leonardo de Moura and Nikolaj Bjørner. Z3: An efficient SMT solver. In *TACAS 2008*, pages 337–340, 2008.
- [EEPT06] Harmut Ehrig, Karsten Ehrig, Ulrike Prange, and Gabriele Taentzer. *Fundamentals of Algebraic Graph Transformation*. Springer, 2006.
- [ISO02] ISO. Information technology—Z formal specification notation—syntax, type system and semantics, 2002. ISO/IEC 13568:2002, Int. Standard.
- [Jac06] Daniel Jackson. *Software Abstractions: logic, lanaguage, and analysis*. MIT Press, 2006.
- [Spi92] J. M. Spivey. *The Z notation: A reference manual*. Prentice-Hall, 1992.

Appendix A

Auxiliary Definitions

This appendix presents the auxiliary definitions that are used to describe the VCL type system presented in chapter 5.

A.1 Environment Operators

Several operators manipulate environments. E_1, E_2 means that two disjoint environments are combined into one. This is defined as set union for each component of the environments being combined:

$$E_1, E_2 = (VE_1 \cup VE_2, SE_1 \cup SE_2, SubE_1 \cup SubE_2) \\ \text{where, } E_1 = (VE_1, SE_1, SubE_1) \wedge E_2 = (VE_2, SE_2, SubE_2)$$

VE_1, VE_2 means that two disjoint variable environments are combined into one. This is defined as set union:

$$VE_1, VE_2 = VE_1 \cup VE_2 \Leftrightarrow \text{dom } VE_1 \cap \text{dom } VE_2 = \emptyset$$

Another operation on variable environments is \bowtie , which merges two variable environments. This requires that if there are identifiers in common in both variable environments, then they must be bound to the same type. This operator is defined as a partial function:

$$_ \bowtie _ : VE \times VE \rightarrow VE$$

This is defined inductively by the following equations:

$$\{\} \bowtie VE = VE \\ (\{id : T\} \cup VE_1) \bowtie VE_2 = VE_1 \bowtie (VE_2 \cup \{Id : T\}) \Leftrightarrow id \notin \text{dom } VE_2 \vee VE_2(Id) = T$$

We define an operator for performing subtractions on variable environments that require that identifiers in common in both variable environments are bound to the same type. This operator is defined as a partial function:

$$_ \boxminus _ : VE \times VE \rightarrow VE$$

This is defined by the following equation:

$$VE_1 \sqcap VE_2 = VE_1 \setminus VE_2 \Leftrightarrow (\forall Id \in (\text{dom } VE_1 \cap \text{dom } VE_2) \bullet VE_1(Id) = VE_2(Id))$$

$E, Id : T$ means that a variable binding is added to an environment. This is defined as:

$$E, Id : T = \begin{cases} (VE \cup \{Id \mapsto T\}, SE, SubE) & \text{If } E = (VE, SE, SubE) \wedge \neg Id \in \text{dom } E.VE \\ \text{undefined} & \text{otherwise} \end{cases}$$

$E, T_1 <: T_2$ means that a subtyping tuple is added to an environment. This is defined as:

$$E, T_1 <: T_2 = (VE, SE, SubE \cup \{T_1 \mapsto T_2\}) \quad \text{where, } E = (VE, SE, SubE)$$

$E, Id \mapsto^{se} (SK, DK, Id, VE)$ means that a set environment binding is added to an environment. This is defined as:

$$E, Id \mapsto^{se} (SK, DK, Id, VE) = \begin{cases} (VE, PE, SE \cup \{(SK, DK, Id, VE)\}, SubE) & \text{If } E = (VE, SE, SubE) \wedge \neg Id \in \text{dom } E.SE \\ \text{undefined} & \text{otherwise} \end{cases}$$

$E \oplus VE$ means that an environment is overridden with a set of variable bindings. This is defined as:

$$E \oplus VE_2 = (VE_1 \oplus VE_2, PE, SE, SubE) \quad \text{where, } E = (VE_1, PE, SE, SubE)$$

A.2 Predicates

$$Acyclic(R) \Leftrightarrow R \in \{rel : X \leftrightarrow X \mid rel^+ \cap id X = \emptyset\}$$

$$IsPEP(PE \overline{PEP}) \Leftrightarrow PE = PEP \wedge (\overline{PE} = \epsilon \vee IsPEP(\overline{PE}))$$

$$IsPEM(PE \overline{PEP}) \Leftrightarrow PE = PEM \wedge (\overline{PE} = \epsilon \vee IsPEM(\overline{PE}))$$

A.3 Auxiliary Functions

A.3.1 Function $getGType$

The function $getGType$ gets the greatest type between two types ordered by the subtyping relation:

$$\begin{aligned} \mathbf{getGType} : E \times Type \times Type &\rightarrow Type \\ \mathbf{getGType}(E, T_1, T_2) &= \begin{cases} T_1 & \text{If } E \vdash T_2 <: T_1 \\ T_2 & \text{If } E \vdash T_1 <: T_2 \\ T_4 & \text{If } (T_1, T_3) \in E.SubE \wedge T_4 = \mathbf{getGType}(E, T_3, T_2) \\ \text{undefined} & \text{otherwise} \end{cases} \\ \mathbf{getGType}(E, \mathbf{Pair}(T_1, T_2), \mathbf{Pair}(T_3, T_4)) &= \mathbf{Pair}(\mathbf{getGType}(E, T_1, T_3), \mathbf{getGType}(E, T_2, T_4)) \end{aligned}$$

A.3.2 Functions producing variable environments (VEs)

The function *getVE* extracts variable environments from set types:

$$\begin{aligned} \mathbf{getVE} &: T \times E \rightarrow VE \\ \mathbf{getVE}(T, E) &= \begin{cases} VE & \text{If } T = \mathbf{Set} \, Id_s \wedge E.SE(Id_s) = (SK, DK, VE) \\ \{\} & \text{otherwise} \end{cases} \end{aligned}$$

The function *consVEs* constructs a pair of variable environments given an optional imports qualifier and a variable environment (*VE*). This function simply makes the given *VE* the first component of the pair if there is an imports qualifier and makes it the second component of the pair otherwise:

$$\begin{aligned} \mathbf{consVEs} &: VE \times \uparrow_{\perp} \rightarrow VE \times VE \\ \mathbf{consVEs}(VE, \uparrow_{\perp}) &= \begin{cases} (VE, \{\}) & \text{if } \uparrow_{\perp} = \uparrow \\ (\{\}, VE) & \text{if } \uparrow_{\perp} = \perp \end{cases} \end{aligned}$$

A.3.3 Function *getDK*

The function *getDK* extracts the definitional kind:

$$\begin{aligned} \mathbf{getDK} &: [\bigcirc] \rightarrow DK \\ \mathbf{getDK}(\bigcirc) &= \mathbf{def} \\ \mathbf{getDK}(\epsilon) &= \mathbf{notDef} \end{aligned}$$

A.3.4 Functions to extract information from ADs

The following functions extract the AD identifier, set identifier and declarations from *ADs*:

$$\begin{aligned} \mathbf{getIdOfAD} &: AD \rightarrow Id \\ \mathbf{getIdOfAD}(\mathbf{AD} \, Id_A [: Id_s] \mathbf{decls} \{ \overline{D} \} \mathbf{pred} \{ \overline{F} \}) &= Id_A \\ \mathbf{getSIdOfAD} &: AD \rightarrow Id_{\perp} \\ \mathbf{getSIdOfAD}(\mathbf{AD} \, Id_A [: Id_s] \mathbf{decls} \{ \overline{D} \} \mathbf{pred} \{ \overline{F} \}) &= Id_s \\ \mathbf{getSIdOfAD}(\mathbf{AD} \, Id_A \mathbf{decls} \{ \overline{D} \} \mathbf{pred} \{ \overline{F} \}) &= \perp \\ \mathbf{getDeclsOfAD} &: AD \rightarrow \overline{D} \\ \mathbf{getDeclsOfAD}(\mathbf{AD} \, Id_A [: Id_s] \mathbf{decls} \{ \overline{D} \} \mathbf{pred} \{ \overline{F} \}) &= \overline{D} \end{aligned}$$

The following functions get the set of ADs that are included in some *AD*:

$$\begin{aligned}
& \mathbf{getDepsOfAD} : AD \times \overline{AD} \times Id_{\perp} \rightarrow \widehat{AD} \\
& \quad getDepsOfAD (AD, \overline{AD}, Id_{s\perp}) = getADsOfDecls (getDeclsOfAD (AD), \overline{AD}, Id_{s\perp}) \\
& \mathbf{getADsOfDecls} : \overline{D} \times \overline{AD} \times Id_{\perp} \rightarrow \widehat{AD} \\
& \quad getADsOfDecls (\epsilon, \overline{AD}, Id_{s\perp}) = \{\} \\
& \quad getADsOfDecls (D \overline{D}, \overline{AD}, Id_{s\perp}) = \\
& \quad \quad getADsOfDecl(D, \overline{AD}, Id_{s\perp}) \cup getADsOfDecls (\overline{D}, \overline{AD}, Id_{s\perp}) \\
& \mathbf{getADsOfDecl} : Decl \times \overline{AD} \times Id_{\perp} \rightarrow \widehat{AD} \\
& \quad getADsOfDecl (DV Id : TD, \overline{AD}, Id_{s\perp}) = \{\} \\
& \quad getADsOfDecl (DF, \overline{AD}, Id_{s\perp}) = getADsOfDF (DF, \overline{AD}, Id_{s\perp}) \\
& \mathbf{getADsOfDF} : DF \times \overline{AD} \times Id_{\perp} \rightarrow \widehat{AD} \\
& \quad getADsOfDF ([\uparrow] \mathbf{assertion} Id_A [\overline{R}], \overline{AD}, Id_{s\perp}) = \{findAD(\overline{AD}, Id_{s\perp}, Id_A)\} \\
& \quad getADsOfDF ([\uparrow] \mathbf{assertion} Id_o.Id_A [\overline{R}], \overline{AD}, Id_{s\perp}) = findLADsWithName(\overline{AD}, Id_A) \\
& \quad getADsOfDF ([\uparrow] \mathbf{assertion} Id_s \rightarrow Id_A [\overline{R}], \overline{AD}, Id_{s\perp}) = \{findAD(\overline{AD}, Id_s, Id_A)\} \\
& \quad getADsOfDF (\neg (DF), \overline{AD}, Id_{s\perp}) = getADsOfDF (DF, \overline{AD}, Id_{s\perp}) \\
& \quad getADsOfDF ((DF_1 \text{ FOp } DF_2), \overline{AD}, Id_{s\perp}) = getADsOfDF (DF_1, \overline{AD}, Id_{s\perp}) \\
& \quad \quad \cup getADsOfDF (DF_2, \overline{AD}, Id_{s\perp}) \\
& \mathbf{getMatchingAD} : AD \times Id \rightarrow \widehat{AD} \\
& \quad getMatchingAD(AD, Id_A) = \begin{cases} \{AD\} & \text{If } getSIdOfAD(AD) \neq \perp \wedge getIdOfAD(AD) = Id_A \\ \{\} & \text{otherwise} \end{cases} \\
& \mathbf{findLADsWithName} : \overline{AD} \times Id \rightarrow \widehat{AD} \\
& \quad findLADsWithName(\epsilon, Id_A) = \{\} \\
& \quad findLADsWithName(AD \overline{AD}, Id_A) = getMatchingAD(AD, Id_A) \cup findLADsWithName(\overline{AD}, Id_A)
\end{aligned}$$

A.3.5 Functions for AD lookup

The following functions look for some AD in a sequence of AD s:

$$\begin{aligned}
& \mathbf{findAD} : \overline{AD} \times Id_{s\perp} \times Id_A \rightarrow AD \\
& \quad findAD(\overline{AD}, \perp, Id_A) = findGblAD(\overline{AD}, Id_A) \\
& \quad findAD(\overline{AD}, Id_s, Id_A) = findLAD(\overline{AD}, Id_s, Id_A) \\
& \mathbf{findGblAD} : \overline{AD} \times Id_A \rightarrow AD \\
& \quad findGblAD(AD, Id_A) = AD \Leftrightarrow getIdOfAD(AD) = Id_A \\
& \quad findGblAD(AD \overline{AD}, Id_A) = AD \Leftrightarrow getIdOfAD(AD) = Id_A \\
& \quad findGblAD(AD \overline{AD}, Id_A) = findGblAD(\overline{AD}, Id_A) \Leftrightarrow getIdOfAD(AD) \neq Id_A \\
& \mathbf{findLAD} : \overline{AD} \times Id_s \times Id_A \rightarrow AD \\
& \quad findLAD(AD, Id_s, Id_A) = AD \Leftrightarrow getSIdOfAD(AD) = Id_s \wedge getIdOfAD(AD) = Id_A \\
& \quad findLAD(AD \overline{AD}, Id_s, Id_A) = AD \Leftrightarrow getSIdOfAD(AD) = Id_s \wedge getIdOfAD(AD) = Id_A \\
& \quad findLAD(AD \overline{AD}, Id_s, Id_A) = findLAD(\overline{AD}, Id_s, Id_A) \\
& \quad \quad \Leftrightarrow getIdOfAD(AD) \neq Id_A \vee getSIdOfAD(AD) \neq Id_s
\end{aligned}$$

A.3.6 Functions for substitutions

The following functions deal with substitutions in variable environments:

$$\begin{aligned}
& \mathbf{doSubs} : VE \times VE \times \overline{R} \rightarrow VE \times VE \\
& \quad doSubs(VE_v, VE_h, \overline{R}) = (applySubs(VE_v, \overline{R}), applySubs(VE_h, \overline{R}))
\end{aligned}$$

applySubs : $VE \times \overline{R} \rightarrow VE$
 $substitute(VE, idn/ido) = \begin{cases} (VE \setminus \{(ido, VE(ido))\}) \cup \{(idn, VE(ido))\} & \text{If } ido \in \text{dom } VE \wedge idn \notin \text{dom } VE \\ undefined & \text{otherwise} \end{cases}$
 $applySubs(VE, \epsilon) = VE$
 $applySubs(VE, R \overline{R}) = applySubs(substitute(VE, R), \overline{R})$

A.3.7 Function *getSIdFrScalarOrCollection*

The following function retrieves a set identifier from types involving set types, which may either denote a scalar or a collection:

getSIdFrTy : $Type \rightarrow Id$
 $getSIdFrTy(\mathbf{Set } Id_s) = Id_s$
 $getSIdFrTy(\mathbf{Pow Set } Id_s) = Id_s$
 $getSIdFrTy(\mathbf{Seq Set } Id_s) = Id_s$
 $getSIdFrTy(\mathbf{Opt Set } Id_s) = Id_s$

Appendix B

Alloy Metamodels

B.1 VCL Common

```
=====
-- Name: 'VCL_Common'
-- Description:
--   + Common entities of VCL ADs and SDs
=====

module VCL_Common

=====
-- Name: 'Name'
-- Description:
--   + Introduces set of labels to be attached to nodes and edges
=====
sig Name {}

=====
-- Name: 'SetElement'
-- Description:
--   + Defines a set element
--   + Either a single object or a pair
=====
abstract sig SetElement {
}

=====
-- Name: 'VCLObject'
-- Description:
--   + A named VCL object
--   + Elements that can be inside a set (either primitive or derived)
=====
sig VCLObject extends SetElement {
  id : Name
}
```

```

}

=====
-- Name: 'Pair'
-- Description:
--   + Represents a pair made of two named objects
=====
sig Pair extends SetElement {
  idElem1 : Name,
  idElem2 : Name
}

=====
-- Name: 'Assertion'
-- Description:
--   + Defines assertions whose symbol is the elongated hexagon.
=====
sig Assertion {
  idAssertion : Name
}

=====
-- Name: 'TypeDesignator'
--
-- Description:
--   + Defines a designator for types.
=====

abstract sig TypeDesignator {
}

=====
-- Name: 'TypeDesignator', ' TypeDesignatorNat'
--
-- Description:
--   + Defines a type designator naturals and integers.
=====
sig TypeDesignatorInt, TypeDesignatorNat extends TypeDesignator {
}

=====
-- Name: 'TypeDesignatorId'
--
-- Description:
--   + Defines a designator of sets with an identifier.
=====
sig TypeDesignatorId extends TypeDesignator {
  setId : Name
}

```

```

=====
-- Name: 'PropEdge'
--
-- Description:
--   + Defines property edges with a source and a target.
=====
abstract sig PropEdge {
  op : EdgeOperator,
  target : Expression,
}

=====
-- Name: 'PropEdgePred'
--
-- Description:
--   + Defines property edges attached to predicate elements.
=====
sig PropEdgePred extends PropEdge {
  unop : lone EdgeOperatorUnary,
  designator : lone Name
}{
  -- 'op' must be a 'EdgeOperatorPred'
  op in EdgeOperatorBin
}

=====
-- Name: 'PropEdgeMod'
--
-- Description:
--   + Defines the property edge modifier that applies some operation to
--     the source.
=====
sig PropEdgeMod extends PropEdge {
}{
  -- 'op' must be a 'EdgeOperatorMod'
  op in EdgeOperatorMod
}

=====
-- Name: 'EdgeOperator'
--
-- Description:
--   + Defines edge operator used in edges.
=====
abstract sig EdgeOperator {
}

=====

```

```

-- Name: 'EdgeOperatorBin'
--
-- Description:
--   + Defines edge operator used in predicate edges.
=====
abstract sig EdgeOperatorBin extends EdgeOperator{
}

=====

-- Name: 'EdgeOperatorMod'
--
-- Description:
--   + Defines edge operator used in modifier edges.
=====
abstract sig EdgeOperatorMod extends EdgeOperator{
}

=====

-- Name: 'EdgeOperatorUnary'
--
-- Description:
--   + Defines edge operator used in modifier edges.
=====
abstract sig EdgeOperatorUnary extends EdgeOperator{
}

=====

-- Name: 'EdgeOperatorEq', 'EdgeOperatorIn', 'EdgeOperatorSubsetEq'
-- 'EdgeOperatorLT', 'EdgeOperatorLEQ', 'EdgeOperatorGT', 'EdgeOperatorGEQ'
--
-- Description:
--   + Defines different kinds of edge operators.
--   + Eq (=), Neq ( ), In ( ), LT, (<), LEQ ( ), GT (>), GEQ ( )
--   + SubsetEq ( )
=====
one sig EdgeOperatorEq,
EdgeOperatorNEq,
EdgeOperatorIn,
EdgeOperatorLT,
EdgeOperatorLEQ,
EdgeOperatorGT,
EdgeOperatorGEQ,
EdgeOperatorSubsetEq
  extends EdgeOperatorBin {
}

=====

-- Name: 'EdgeOperatorDRES', 'EdgeOperatorRRES'
--

```

```

-- Description:
--   + Edge Operators used in property edge modifiers.
--   + DRES ( , domain restriction), and RRES ( , range restriction)
--   + DSUB ( , domain subtraction) and RSUB ( , range subtraction)
=====

one sig EdgeOperatorDRES,
EdgeOperatorRRES,
      EdgeOperatorDSUB,
EdgeOperatorRSUB
extends EdgeOperatorMod {
}

=====

-- Name: 'EdgeOperatorCARD', 'EdgeOperatorTHE'
-- Description:
--   + Unary edge operator used in predicate property edges
--   + CARD (#, cardinality)
--   + THE ( , the)
=====

one sig EdgeOperatorCARD, EdgeOperatorTHE
      extends EdgeOperatorUnary {
}

=====

-- Name: 'Num'
--
-- Description:
--   + String representing natural numbers.
=====

sig Num {}

=====

-- Name: 'Expression'
--
-- Description:
--   + Defines expressions associated with property edges.
=====

abstract sig Expression {
}

=====

-- Name: 'FreeExpression'
--
-- Description:
--   + Defines a free (editable) expression.
=====

abstract sig FreeExpression extends Expression {
}

```

```

=====
-- Name: 'FreeExpId'
--
-- Description:
--   + Defines object expressions comprising an identifier (a name).
=====

sig FreeExpId extends FreeExpression {
  oid : Name,
  pkgId : lone Name,
}

=====
-- Name: 'FreeExpDot'
--
-- Description:
--   + Defines expressions that access the state of objects.
=====
abstract sig FreeExpDot extends FreeExpression {
  oid : Name, -- Identifier of the object
  propId : Name -- Identifier of the property
}

=====
-- Name: 'FreeExpNum'
--
-- Description:
--   + Defines expressions comprising a number.
=====

sig FreeExpNum extends FreeExpression {
  num : Num
}

=====
-- Name: 'FreeExpUMinus'
--
-- Description:
--   + Defines unary minus expression (-e).
=====
sig FreeExpUMinus extends FreeExpression {
  e : FreeExpression
}

=====
-- Name: 'FreeExpBin'
--

```

```

-- Description:
--   + Defines expressions that can be combined with binary operators.
=====
abstract sig FreeExpBin extends FreeExpression {
  e1, e2 : FreeExpression,
  op : FreeExpBinOp
}{
  e1 != e2
}

=====

-- Name: 'FreeExpPar'
--
-- Description:
--   + Defines expressions that can be placed within parenthesis.
=====
abstract sig FreeExpPar extends FreeExpression {
  e : FreeExpression
}

=====

-- Name: 'FreeExpBinOp'
--
-- Description:
--   + Infix operators for sum (+), subtraction (-), product (*), div (/).
=====
abstract sig FreeExpBinOp {}

one sig FreeExpBinOpPlus,
  FreeExpBinOpMinus,
  FreeExpBinOpTimes,
  FreeExpBinOpDiv extends FreeExpBinOp {}

=====

-- Name: 'SetExpression'
--
-- Description:
--   + Defines a set expression.
=====
abstract sig SetExpression extends Expression {
}

=====

-- Name: 'SetExpressionID'
--
-- Description:
--   + Defines a set expression defined using a type designator.
=====
sig SetExpressionID extends SetExpression {

```

```

    bd : TypeDesignator
}

=====
-- Name: 'SetExpressionEmpty'
--
-- Description:
--   + Defines a set that is shaded to represent the empty set.
=====
sig SetExpressionEmpty extends SetExpression {
}

=====
-- Name: 'SetExpressionCard'
--
-- Description:
--   + Defines a set with a cardinality unary operator attached.
=====
sig SetExpressionCard extends SetExpression {
    setExp : SetExpression
}

=====
-- Name: 'SetDef'
--
-- Description:
--   + Defines a set definition (symbol ).
=====
sig SetDef {
    bdop      : SetDefOp, -- optional blob def operator
    insideExp : SetInsideExpression
}

=====
-- Name: 'SetDefOp'
--
-- Description:
--   + Defines set def operators
--   + Domain operator is represented as symbol
--   + Range operator is represented as symbol
--   + None represents no symbol
--   + Union operator is represented as symbol
--   + Intersection operator is represented as symbol
--   + Cross product operator is represented as symbol
--   + Relation composition operator is represented by symbol
--   + Set difference operator is represented as symbol
=====
abstract sig SetDefOp {

```



```

}

one sig SetDefOpDomain,
    SetDefOpRange,
    SetDefOPNone,
    SetDefOpUnion,
    SetDefOpIntersection,
    SetDefOpCrossProduct,
    SetDefOpSetMinus,
    SetDefOpRelComp
    extends SetDefOp {
}

=====
-- Name: 'SetExpressionDef'
--
-- Description:
--   + Defines a set expression defined using a set definition.
=====

sig SetExpressionDef extends SetExpression {
    def : SetDef
}

=====
-- Name: 'SetInsideExpression'
--
-- Description:
--   + Expression inside the set definition
=====
abstract sig SetInsideExpression {
}

=====
-- Name: 'InsideExpB1Ds'
--
-- Description:
--   + Expression inside the set def
=====
sig InsideExpB1Ds extends SetInsideExpression {
    blobDefs : seq SetDef
}

=====
-- Name: 'InsideDef'
--
-- Description:

```

```

--      + Definition of the blob def
--      + Either a constrained blob or a a set extension
=====
abstract sig InsideDef extends SetInsideExpression {
}

=====

-- Name: 'ConstrainedSet'
--
-- Description:
--      + Defines a set with restrictions (constraints).
=====

sig ConstrainedSet extends InsideDef {
  bd  : TypeDesignator,
  pes : seq PropEdge -- 0 or more predicate property edges
}

fact PropEdgesOfConstrainedSetAreOfSomeKind {
  all be :ConstrainedSet |
    all disj pe1, pe2 : univ.(be.pes) |
      pe1+pe2 in PropEdgePred || pe1+pe2 in PropEdgeMod
}

=====

-- Name: 'SetExtension'
--
-- Description:
--      + Defines a set extensionally by listing its members.
=====

sig SetExtension extends SetInsideExpression {
  elems : some SetElement
}

```

B.2 Bool Module

```

=====

-- Name: 'Bool'
--
-- Description:
--      + Signature of booleans: 'True' or 'False'.
=====

abstract sig Bool {}

one sig True, False extends Bool {}

```

B.3 VCL Structural Diagrams

```
=====
-- Name: 'VCL_SD'
-- Description:
--   + Defines meta-model of VCL structural diagrams (SDs).
=====
module VCL_SD

open VCL_Common as c
open Bool

-- Name: 'Mult' (Multiplicity)
--
-- Description:
--   + Defines what a multiplicity is.
--   + Multiplicities are attached to ends of edges.
-- Details:
--   + There are the following kinds of multiplicity: one, optional (0..1),
--   many (0..*), one or many (1..*), range (n1..n2) and sequence.
--   + Multiplicities of kind range have a lower and an upper bound.
=====
abstract sig Mult {}

one sig MOne, MOpt, MMany, MOneOrMany, MSeq extends Mult {}

one sig MStar {}

sig MRange extends Mult {
  -- lower and upper bounds for 'range' multiplicities.
  lb : Int,
  ub : (Int+MStar)
}{
  -- lower and upper bounds must be greater or equal than 0
  -- and 'ub' greater or equal than 'lb'.
  lb >= 0 && (ub = MStar || ub >= lb)
}

-- Name: 'SDElem'
--
-- Description:
--   + Introduces the labelled structural diagram element.
--   + To be extended by 'Set', 'Object', 'Edge'.
=====
abstract sig SDElem {
  name : Name -- a modelling element has a name (a label).
}
```

```

=====
-- Name: 'Constant'
--
-- Description:
--   + Represents constants. A constant has a type (field 'type').
--   + Constants can be 'local' or 'global'.
--   + A constant definition has a type
=====

sig Constant extends SDElem {
  type      : lone Name
}

=====
-- Name: 'RelEdge' (Relational Edge)
--
-- Description:
--   + Set relational edges are binary edges connecting sets.
--   + They have multiplicities at each end of edge.
=====

sig RelEdge extends SDElem {
  source, target : Set,
  sourceMult, targetMult : Mult,
}{
  -- Relation edges cannot have multiplicities of type sequence
  not (sourceMult+targetMult) in MSeq
}

=====
-- Name: 'Set' (Set Definitions)
--
-- Description:
--   + Defines a global set definition.
--   + It's characterised by inside property.
--
=====

abstract sig Set extends SDElem {
}

=====
-- Name: 'IntSet' (Integer Set)
--
-- Description:
--   + Defines a set representing the integers

```

```

=====
one sig IntBlob extends Set {}

=====
-- Name: 'NatSet' (Natural numbers Set)
--
-- Description:
--   + Defines a set representing the natural numbers
=====
one sig NatSet extends Set {}

abstract sig SetKind {}

=====
-- Name: 'Value', 'Class'
--
-- Description:
--   + Defines two set kinds: 'value' and 'class'.
=====

one sig Value, Class extends SetKind {}

=====
-- Name: 'SetDefObject'
--
-- Description:
--   + An object that can be inside a primitive set.
=====

sig SetDefObject {
  objName : Name
}

=====
-- Name: 'PrimarySet'
-- Description:
--   + Defines a primary set
--   + A Primary set can have sets and objects inside.
=====
sig PrimarySet extends Set {
  kind          : SetKind,
  lProps        : set PropEdgeDef,
  hasInsideSet  : set PrimarySet,
  isDefSet      : Bool, -- (symbol if 'True')
  hasInside0    : set SetDefObject,
  lInvariants   : set Assertion,
  lConstants    : set Constant,
}

```

```

--
-- The following defines what it means for VCL structures to be well-formed
-- regarding the 'inside' property

--
-- The graph representing the 'inside' relation should be acyclic.
fact acyclicInside {
  no ^(hasInsideSet) & iden
}

--
-- An object should be in at most one set (the inverse of the relation is a partial function)
fact setInAtMostOneBlob {
  all s : PrimarySet | lone s.~hasInsideSet
}

--
-- An object should be in at most one set (the inverse of the relation is a partial function)
fact objInAtMostOneBlob {
  all n : SetDefObject | lone n.~hasInside0
}

--
-- Each 'Set' has its own set of local invariants.
-- Or local invariants are not shared.
fact LInvariantsNotShared {
  all c : Assertion | (some lInvariants.c)
    => one lInvariants.c
}

--
-- Each 'Set' has its own set of local constants
-- Or local constants are not shared.
fact LConstantsNotShared {
  all c : Constant | (some lConstants.c)
    => one lConstants.c
}

--
-- Definitional sets must have things inside.
fact DefSetsHasThingsInside {
  all b : isDefSet.True | #b.hasInside0 > 0 || #b.hasInsideSet > 0
}

--
-- Each class set can contain other classes obly
-- and they can be inside of class sets only.
fact ClSetHasClSetsInside {
  all b : PrimarySet | b.kind = Class
    => (b.hasInsideSet) in kind.Class && hasInsideSet.b in kind.Class
}

```

```

}

-----
-- Name: 'PropEdgeDef' (Property Edge Definition)
-- Description:
--   + Defines properties of sets.
--   + Relates one blob (having property) to another (type of property).
--   + A property edge has a 'Set' as target.
--   + A property edge may have a multiplicity.
--
--   -----          0..*-----
--   |PropEdge|----->|Set   |
--   -----          target -----
-----

sig PropEdgeDef extends SDElem {
  peTarget : Set,
  mult      : Mult
}
{
  -- a PropEdgeDef cannot have its blob or his inside blobs as target
  not (peTarget in ((this.lProps) + (this.lProps).^(hasInsideSet)))
}

--
-- Each 'Set' has its own set of property edge definitions
-- Or property edges are not shared. All property edges belong to some set
fact propEdgesNotSharedAndBelongToSomeSet {
  all pe : PropEdgeDef | one lProps.pe
}

fun nameOf (elem : SDElem + Assertion) : Name {
  elem in SDElem implies elem.name else elem.idAssertion
}

--
-- Local Names in the scope of a 'Set' must be unique
--
fact LocalNamesAreUnique {
  all s : Set |
  all e1, e2 : (s.lConstants+s.lInvariants+s.lProps+(s.hasInside0))
  | nameOf [e1] = nameOf [e2]
  => e1 = e2
}

--
-- All global names must be unique
fact GblNamesAreUnique {
  all e1, e2 :
    (Set+(Assertion-(PrimarySet.lInvariants)))+

```

```

        RelEdge+(Constant-(PrimarySet.lConstants)))
    | nameOf[e1] = nameOf[e2] implies e1 = e2
}

=====
-- Name: 'DerivedSet'
--
-- Description:
--   + Defines a derived set
--   + Derived sets make use of symbol ' '
=====
sig DerivedSet extends Set {
    definition : SetDef
}

=====
-- Name: 'SDDiag'
--
-- Description:
--   + Defines a structural diagram
=====
sig SDDiag {
    sdelems : set SDElem,
    invs : set Assertion
}

```

B.4 VCL Assertion Diagrams

```

=====
-- Name: 'VCL_AD'
--
-- Description:
--   + Module defining the meta-model of VCL assertion diagrams.
=====

open VCL_Common as c
open Bool

=====
-- Name: 'Decl'
--
-- Description:
--   + Defines a declaration of an assertion diagram.
=====
abstract sig Decl {
}

```



```

=====
-- Name: 'VarDecl'
--
-- Description:
--   + Defines a typed variable declaration of AD or CD.
--   + A typed variable declaration has a name, type and hidden status
--   + In EMF metamodel 'dNames' is just a string (to be parsed by type-checker)
=====
abstract sig VarDecl extends Decl {
  dNames : set Name, -- set of declaration names separated by commas
  dTy : TypeDesignator, // Type of declaration
  isHidden : Bool, // Indicates whether the variable is hidden or not
}

=====
-- Name: 'DeclObj'
--
-- Description:
--   + Defines declarations of objects.
--   + Declarations of objects are represented as objects (rectangles).
--   + field optional indicates whether declaration is optional or not
--   + If optional is true, then '?' precedes the object's type.
=====

sig DeclObj extends VarDecl{
  optional : Bool
}

=====
-- Name: 'DeclSet'
--
-- Description:
--   + Defines declarations of sets.
--   + Sets are represented as blobs (ovals); they include the word "SET" on
--   top-left corner
=====

sig DeclSet extends VarDecl {
}

=====
-- Name: 'DeclSeq'
--
-- Description:
--   + Defines declarations of sequence.
--   + Sequences are represented as blobs (ovals); they include the word
--   "SEQUENCE" on top-left corner
=====

```

```

sig DeclSeq extends VarDecl {
}

=====
-- Name: 'DeclFormula'
--
-- Description:
--   + Defines a declaration reference formula.
--   + This enables declaration references (either assertions or contracts)
--     to be combined using logical operators.
=====

abstract sig DeclFormula extends Decl {
}

=====
-- Name: 'RenamingExp'
--
-- Description:
--   + Defines a renaming expression, denoted in logic as [u/y]
--     where expression u denoted as the substitution for variable y.
=====
sig RenamingExp {
  subExp : Name, -- Substituting expression
  varToSub : Name -- Variable to substitute
}

=====
-- Name: 'DeclFormulaAtom'
--
-- Description:
--   + A declarations formula atom holds represents references to assertions or contracts
--   + The import is represented by the symbol '↑'
--   + Optional 'callObj' indicates a call a local operation on an object
--     represented as "a.op".
--   + Optional field 'origin' indicates origin of the operation (blob or package).
--   + Optional owning set indicates set of local contract or assertion
--   + Renaming expressions represented as '[t/x,u/y]'. In Ecore,
--     'RenamingExp' is just a String.
=====

abstract sig DeclFormulaAtom extends DeclFormula {
  refId      : Name, -- Name of assertion or contract
  owningSet  : lone Name, -- Id of set that owns local assertion or contract
  callObj    : lone Name, -- Id of obj on which local assertion or contract is called
  origin     : lone Name, -- optional originating package
  import     : Bool,      -- Whether import symbol is present or not
  renameExp  : set RenamingExp -- a set of renaming expressions
}

```

```

=====
-- Name: 'DeclAssertion'
--
-- Description:
--   + Represents an assertion reference of a declarations formula
=====
sig DeclAssertion extends DeclFormulaAtom {
}

=====
-- Name: 'FormulaOp'
--
-- Description:
--   + Defines a formula operator.
=====
abstract sig FormulaOp {
}

=====
-- Name: FImplies, FAnd, FOr, FEquiv
--
-- Description:
--   + Defines formula operators for implication (), conjunction (),
--   disjunction (), equivalence (), negation ( $\neg$ ),
--   + and sequential composition ()
=====
one sig FImplies, FAnd, FOr, FEquiv, FNot, FSComp extends FormulaOp {
}

=====
-- Name: 'DeclFormulaNAry'
--
-- Description:
--   + Defines a declaration binary formula
--   + This supports the logical operators ,
=====
sig DeclFormulaNAr extends DeclFormula {
  dfrmls : DeclFormula,
  dfop   : FormulaOp
}{
  all df1, df2 : dfrmls | df1 != df2
  dfop != FSComp
  dfop in FAnd+FOr implies #dfrmls >= 2
  dfop in FNot implies #dfrmls = 1
  dfop in FImplies+FEquiv implies #dfrmls = 2
}

```

```

=====
-- Name: 'FormulaSource'
--
-- Description:
--   + Defines the source of an arrows formula
--   + It can either be: obj, blob or pair
=====
abstract sig FormulaSource {
}

=====
-- Name: 'FormulaSourceElement'
--
-- Description:
--   + Defines source formula of type object
--   + 'elem' indicates the 'SetElement' either object or pair
=====
sig FormulaSourceElem extends FormulaSource {
  elem : SetElement
}

=====
-- Name: 'FormulaSourceSet'
--
-- Description:
--   + Defines source formula of type set
=====
abstract sig FormulaSourceSet extends FormulaSource {
}

=====
-- Name: 'FormulaSourceSetId'
--
-- Description:
--   + Defines source formula of type blob identifier
--   + 'bId' indicates identifier of the set
=====
sig FormulaSourceSetId extends FormulaSourceSet {
  bId : Name
}

=====
-- Name: 'FormulaSourceSetDef'
--
-- Description:
--   + Defines source formula of type set definition
--   + 'blDef' holds set definition
=====
sig FormulaSourceSetDef extends FormulaSourceSet {

```

```

    blDef : SetDef
}

=====
-- Name: 'FormulaSourceUOp'
--
-- Description:
--   + Defines a unary Formula operator for a formula source.
=====
abstract sig FormulaSourceUOp {
}

=====
-- Name: FSBCardinality, FSBDom, FSBRan
--
-- Description:
--   + Symbol of Formula source operator cardinality is #
--   + Symbol of Formula source operator domain is ' '
--   + Symbol of Formula source operator range is ' '
--   + Symbol of Formula source operator the is ' '
=====
one sig FSBCardinality, FSBDom, FSBRan , FSBThe
extends FormulaSourceUOp {
}

=====
-- Name: 'FormulaSourceUnary'
--
-- Description:
--   + Defines source formula with unary operator
--   + Let '0' be a blob, this construction is expressed as # [0]
=====
sig FormulaSourceUnary extends FormulaSource {
  operator : FormulaSourceUOp,
  frmlSrc  : FormulaSource
}

=====
-- Name: 'AD'
--
-- Description:
--   + Defines what an assertion diagram is.
=====

abstract sig AD {
  aName      : Name,
  declarations : set Decl,
  predicate   : set Formula
}

```

```

}

=====
-- Name: 'Formula'
--
-- Description:
--   + Defines a Formula.
=====
abstract sig Formula {
}

=====
-- Name: 'FormulaNAry'
--
-- Description:
--   + Defines an n-ary Formula
=====
sig FormulaNAry extends Formula {
  frmls : set Formula,
  operator : FormulaOp
}{
  all f1, f2 : frmls | f1 != f2
  operator != FSComp
  operator in FAnd+FOr implies #frmls >= 2
  operator in FNot implies #frmls = 1
  operator in FImplies+FEquiv implies #frmls = 2
}

=====
-- Name: 'QFormula' (Quantified formula)
--
-- Description:
--   + Defines a quantified Formula.
--   + Includes a set of variable declarations and one formula
=====
sig QFormula extends Formula {
  decls : seq QDecl,
  frml : Formula
}{
  -- all elements in the sequence must be distinct
  not decls.hasDups
}

=====
-- Name: 'QDecl' (Quantified Declaration)
--
-- Description:

```

```

--      + Defines a quantified declaration.
--      + Includes a set of variable declarations and one variable kind
=====
sig QDecl {
  qkind : QuantifierKind, -- quantifier kind
  vars  : set VarDecl    -- variable declarations
}{
  -- Vars are distinct
  all v1, v2 : vars | v1 != v2
  -- hidden variables not allowed in quantified formulas
  all v : vars | v.isHidden = False
}

=====
-- Name: 'QuantifierKind'
--
-- Description:
--      + Defines the kind of quantifier: forall or exists.
=====
abstract sig QuantifierKind {
}

=====
-- Name: 'QForAll'
--
-- Description:
--      + Defines the 'forall' (or universal) quantifier kind
--      + Represented in terms of concrete syntax by symbol classic symbol ' '
=====
one sig QForAll extends QuantifierKind {
}

=====
-- Name: 'QExists'
--
-- Description:
--      + Defines the 'exists' quantifier kind
--      + Represented in terms of concrete syntax by symbol classic symbol ' '
=====
one sig QExists extends QuantifierKind {
}

=====
-- Name: 'ArrowsFormula'
--
-- Description:
--      + Defines an arrows formula
--      + Made of predicate property edges

```

```

--      + With a source, which can either be: obj, blob or pair
=====
sig ArrowsFormula extends Formula {
  source : FormulaSource,
  pes : some PropEdgePred
}

=====

-- Name: 'SetFormula'
--
-- Description:
--      + Defines a 'Set' formula.
=====
abstract sig SetFormula extends Formula {
}

=====

-- Name: 'SetFormulaDef'
--
-- Description:
--      + Defines a 'Set' formula using a set definition (symbol )
=====
sig SetFormulaDef extends SetFormula {
  shaded : Bool, -- set may be shaded to mean empty set
  bid : lone TypeDesignator, -- optional set designator
  bdef : SetDef -- set definition
}

=====

-- Name: 'SetFormulaSubset'
--
-- Description:
--      + Defines a 'Set' formula defined using a subset definition.
=====
sig SetFormulaSubset extends SetFormula {
  bid : TypeDesignator,
  hasInside : SetExpression
}

=====

-- Name: 'SetFormulaShaded'
--
-- Description:
--      + Defines a 'Set' formula defined using shading.
=====
sig SetFormulaShaded extends SetFormula {
  bid : TypeDesignator
}

```


Appendix C

Z3 Proofs

C.1 Common

C.1.1 Z3 Encoding

```
(set-option :mbqi true)
(set-option :macro-finder true)
(set-option :pull-nested-quantifiers true)
(set-option :produce-unsat-cores true)
(set-option :produce-models true)

(declare-sort V_MM)
(declare-sort E_MM)

(declare-sort V_G)
(declare-sort E_G)

(declare-const MM_Name          V_MM)
(declare-const MM_Num           V_MM)
(declare-const MM_Assertion     V_MM)
(declare-const MM_VCLObj       V_MM)
(declare-const MM_Pair          V_MM)
(declare-const MM_SetElement    V_MM)
(declare-const MM_InsideDef     V_MM)
(declare-const MM_SetExtension  V_MM)
(declare-const MM_ConstrainedSet V_MM)
(declare-const MM_SetInsideExpression V_MM)
(declare-const MM_InsideExpSDs  V_MM)
(declare-const MM_SetDef        V_MM)
;; SetDefOp
(declare-const MM_SetDefOp      V_MM)
(declare-const MM_SOp_Domain    V_MM)
(declare-const MM_SOp_Range     V_MM)
(declare-const MM_SOp_Union     V_MM)
(declare-const MM_SOp_Intersection V_MM)
```

```

(declare-const MM_SOp_CrossProduct V_MM)
(declare-const MM_SOp_SetMinus V_MM)
(declare-const MM_SOp_RelComp V_MM)
(declare-const MM_SOp_None V_MM)
;; Type Designator
(declare-const MM_TypeDesignator V_MM)
(declare-const MM_TypeDesignatorNat V_MM)
(declare-const MM_TypeDesignatorInt V_MM)
(declare-const MM_TypeDesignatorId V_MM)
;; FreeExpression
(declare-const MM_FreeExpression V_MM)
(declare-const MM_FreeExpId V_MM)
(declare-const MM_FreeExpNum V_MM)
(declare-const MM_FreeExpUMinus V_MM)
(declare-const MM_FreeExpPar V_MM)
(declare-const MM_FreeExpDot V_MM)
(declare-const MM_FreeExpBin V_MM)
;; FreeExpBinOp
(declare-const MM_FreeExpBinOp V_MM)
(declare-const MM_FreeExpBinOp_Plus V_MM)
(declare-const MM_FreeExpBinOp_Minus V_MM)
(declare-const MM_FreeExpBinOp_Times V_MM)
(declare-const MM_FreeExpBinOp_Div V_MM)
;; SetExpression
(declare-const MM_SetExpression V_MM)
(declare-const MM_SetExpressionCard V_MM)
(declare-const MM_SetExpressionId V_MM)
(declare-const MM_SetExpressionEmpty V_MM)
(declare-const MM_SetExpressionDef V_MM)
;; Expression
(declare-const MM_Expression V_MM)
;; PropEdge
(declare-const MM_PropEdge V_MM)
(declare-const MM_PropEdgePred V_MM)
(declare-const MM_PropEdgeMod V_MM)
(declare-const MM_EdgeOperatorBin V_MM)
(declare-const MM_EdgeOperatorMod V_MM)
(declare-const MM_EdgeOperatorUn V_MM)
;; EdgeOperatorUnary
(declare-const MM_UOpCard V_MM)
(declare-const MM_UOpThe V_MM)
(declare-const MM_UOpNone V_MM)
;; EdgeOperatorBin
(declare-const MM_BOpEQ V_MM)
(declare-const MM_BOpNEQ V_MM)
(declare-const MM_BOpIN V_MM)
(declare-const MM_BOpLT V_MM)
(declare-const MM_BOpLEQ V_MM)
(declare-const MM_BOpGT V_MM)

```

```

(declare-const MM_BOpGEQ          V_MM)
(declare-const MM_BOpSubsetEQ     V_MM)
;; EdgeOperatorMod
(declare-const MM_MOpDRES         V_MM)
(declare-const MM_MOpRRES         V_MM)
(declare-const MM_MOpDSUB         V_MM)
(declare-const MM_MOpRSUB         V_MM)
;; Special 'Null' constant to check totality
(declare-const MM_Null            V_MM)

(declare-const G_Num              V_G)
(declare-const G_Id               V_G)
;; TD
(declare-const G_TD               V_G)
(declare-const G_TD_Int           V_G)
(declare-const G_TD_Nat           V_G)
(declare-const G_TD_Id            V_G)
;; A, O, P, SE
(declare-const G_A                V_G)
(declare-const G_O                V_G)
(declare-const G_P                V_G)
(declare-const G_SE               V_G)
;; PE
(declare-const G_PE               V_G)
(declare-const G_PEP              V_G)
(declare-const G_PEM              V_G)
;; UEOp
(declare-const G_UEOp             V_G)
(declare-const G_UEOp_Card        V_G)
(declare-const G_UEOp_The         V_G)
(declare-const G_UEOp_None        V_G)
;; BEOp
(declare-const G_BEOp             V_G)
(declare-const G_BEOp_EQ          V_G)
(declare-const G_BEOp_NEQ         V_G)
(declare-const G_BEOp_IN          V_G)
(declare-const G_BEOp_LT          V_G)
(declare-const G_BEOp_LEQ         V_G)
(declare-const G_BEOp_GT          V_G)
(declare-const G_BEOp_GEQ         V_G)
(declare-const G_BEOp_SubsetEQ    V_G)
;; MOp
(declare-const G_MOp              V_G)
(declare-const G_MOp_DRES         V_G)
(declare-const G_MOp_RRES         V_G)
(declare-const G_MOp_DSUB         V_G)
(declare-const G_MOp_RSUB         V_G)
;; TExp
(declare-const G_TExp             V_G)

```

```

;; FExp
(declare-const G_FExp V_G)
(declare-const G_FExpId V_G)
(declare-const G_FExpDot V_G)
(declare-const G_FExpNum V_G)
(declare-const G_FExpUMinus V_G)
(declare-const G_FExpPar V_G)
(declare-const G_FExpBin V_G)
;; FEOp
(declare-const G_FEOp V_G)
(declare-const G_FEOp_Plus V_G)
(declare-const G_FEOp_Minus V_G)
(declare-const G_FEOp_Times V_G)
(declare-const G_FEOp_Div V_G)
;; SExp
(declare-const G_SExp V_G)
(declare-const G_SExpTD V_G)
(declare-const G_SExpSDef V_G)
(declare-const G_SExpEmpty V_G)
(declare-const G_SExpCard V_G)
;; IDef
(declare-const G_IDef V_G)
(declare-const G_IDef_SExt V_G)
(declare-const G_IDef_CntSet V_G)
;; IExp
(declare-const G_IExp V_G)
(declare-const G_IExp_SDs V_G)
(declare-const G_IExp_IDef V_G)
(declare-const G_SDef V_G)
;; SOp
(declare-const G_SOp V_G)
(declare-const G_SOp_Domain V_G)
(declare-const G_SOp_Range V_G)
(declare-const G_SOp_Union V_G)
(declare-const G_SOp_Intersection V_G)
(declare-const G_SOp_CrossProduct V_G)
(declare-const G_SOp_SetMinus V_G)
(declare-const G_SOp_RelComp V_G)
(declare-const G_SOp_None V_G)
;; Special 'Null' constant to check totality
(declare-const G_Null V_G)

; Assertion, VCLObj, Pair, SetElement
(declare-const MM_EAssertion_Id E_MM)
(declare-const MM_EVCLObj_Id E_MM)
(declare-const MM_EIVCLObj E_MM)
(declare-const MM_EPair_Id1 E_MM)
(declare-const MM_EPair_Id2 E_MM)
(declare-const MM_EIPair E_MM)

```

```

(declare-const MM_ESetExtension_Elems      E_MM)
(declare-const MM_EConstrainedSet_Design   E_MM)
(declare-const MM_EConstrainedSet_PropEdge E_MM)
(declare-const MM_EISetExtension           E_MM)
(declare-const MM_EIConstrainedSet         E_MM)
(declare-const MM_EIInsideDef              E_MM)
(declare-const MM_EIInsideExpSDs          E_MM)
(declare-const MM_ESetDef_insideExp       E_MM)
(declare-const MM_ESetDef_sdop             E_MM)
(declare-const MM_EIInsideExpSDs_setDefs   E_MM)
;; SetDefOp
(declare-const MM_EISOp_Domain             E_MM)
(declare-const MM_EISOp_Range              E_MM)
(declare-const MM_EISOp_Union              E_MM)
(declare-const MM_EISOp_Intersection       E_MM)
(declare-const MM_EISOp_CrossProduct       E_MM)
(declare-const MM_EISOp_SetMinus           E_MM)
(declare-const MM_EISOp_RelComp            E_MM)
(declare-const MM_EISOp_None               E_MM)
;; Type Designator
(declare-const MM_ETypeDesignatorId        E_MM)
(declare-const MM_EITypeDesignatorId       E_MM)
(declare-const MM_EITypeDesignatorNat      E_MM)
(declare-const MM_EITypeDesignatorInt      E_MM)
;; FreeExpression
(declare-const MM_EFreeExpNum              E_MM)
(declare-const MM_EIFreeExpNum             E_MM)
(declare-const MM_EFreeExpId               E_MM)
(declare-const MM_EIFreeExpId              E_MM)
(declare-const MM_EFreeExpUMinus           E_MM)
(declare-const MM_EIFreeExpUMinus          E_MM)
(declare-const MM_EFreeExpPar              E_MM)
(declare-const MM_EIFreeExpPar             E_MM)
(declare-const MM_EFreeExpDotId            E_MM)
(declare-const MM_EFreeExpDotPropId        E_MM)
(declare-const MM_EIFreeExpDot             E_MM)
(declare-const MM_EFreeExpBinExp1          E_MM)
(declare-const MM_EFreeExpBinExp2          E_MM)
(declare-const MM_EFreeExpBinOp            E_MM)
(declare-const MM_EIFreeExpBinExp          E_MM)
;; FreeExpBinOp
(declare-const MM_EIFreeExpBinOp_Plus      E_MM)
(declare-const MM_EIFreeExpBinOp_Minus     E_MM)
(declare-const MM_EIFreeExpBinOp_Times     E_MM)
(declare-const MM_EIFreeExpBinOp_Div       E_MM)
;; SetExpression
(declare-const MM_ESetExpressionCard       E_MM)
(declare-const MM_EISetExpressionCard      E_MM)
(declare-const MM_ESetExpressionId         E_MM)

```

```

(declare-const MM_EISetExpressionId      E_MM)
(declare-const MM_EISetExpressionEmpty    E_MM)
(declare-const MM_ESetExpressionDef       E_MM)
(declare-const MM_EISetExpressionDef      E_MM)
;; Expression
(declare-const MM_EISetExpression        E_MM)
(declare-const MM_EIFreeExp              E_MM)
;; PropEdge
(declare-const MM_EPropEdgeTarget        E_MM)
(declare-const MM_EPropEdgePredBOp      E_MM)
(declare-const MM_EPropEdgePredName     E_MM)
(declare-const MM_EPropEdgePredUOp      E_MM)
(declare-const MM_EPropEdgeModMOp       E_MM)
(declare-const MM_EIPropEdgeMod         E_MM)
(declare-const MM_EIPropEdgePred        E_MM)
;; EdgeOperatorUnary
(declare-const MM_EIUOp_Card             E_MM)
(declare-const MM_EIUOp_The              E_MM)
(declare-const MM_EIUOp_None            E_MM)
;; EdgeOperatorBin
(declare-const MM_EIBOp_EQ               E_MM)
(declare-const MM_EIBOp_NEQ             E_MM)
(declare-const MM_EIBOp_In              E_MM)
(declare-const MM_EIBOp_LT              E_MM)
(declare-const MM_EIBOp_LEQ             E_MM)
(declare-const MM_EIBOp_GT              E_MM)
(declare-const MM_EIBOp_GEQ             E_MM)
(declare-const MM_EIBOp_SubsetEQ        E_MM)
;; EdgeOperatorMod
(declare-const MM_EIMOp_DRES             E_MM)
(declare-const MM_EIMOp_RRES            E_MM)
(declare-const MM_EIMOp_DSUB            E_MM)
(declare-const MM_EIMOp_RSUB            E_MM)
;; Special 'Null' constant to check totality
(declare-const MM_ENull                  E_MM)

;; TD
(declare-const G_E_TD_Id                 E_G)
(declare-const G_E_TD_Def_Id             E_G)
(declare-const G_E_TD_Def_Nat            E_G)
(declare-const G_E_TD_Def_Int            E_G)
;; A, O, P, SE
(declare-const G_E_A_Id                  E_G)
(declare-const G_E_SE_Def_O              E_G)
(declare-const G_E_SE_Def_P              E_G)
(declare-const G_E_O_Id                  E_G)
(declare-const G_E_P_Id_1                E_G)
(declare-const G_E_P_Id_2                E_G)
;; PE

```

```

(declare-const G_E_PE_TExp          E_G)
(declare-const G_E_PE_PEP           E_G)
(declare-const G_E_PE_PEM           E_G)
(declare-const G_E_PEP_UEOp         E_G)
(declare-const G_E_PEP_Id           E_G)
(declare-const G_E_PEP_BEOp         E_G)
(declare-const G_E_PEM_MOp          E_G)
;; UEOp
(declare-const G_E_UEOp_Def_Card     E_G)
(declare-const G_E_UEOp_Def_The      E_G)
(declare-const G_E_UEOp_Def_None     E_G)
;; BEOp
(declare-const G_E_BEOp_Def_Eq       E_G)
(declare-const G_E_BEOp_Def_Neq      E_G)
(declare-const G_E_BEOp_Def_In       E_G)
(declare-const G_E_BEOp_Def_LT       E_G)
(declare-const G_E_BEOp_Def_LEQ      E_G)
(declare-const G_E_BEOp_Def_GT       E_G)
(declare-const G_E_BEOp_Def_GEQ      E_G)
(declare-const G_E_BEOp_Def_SUBSETEQ E_G)
;; MOp
(declare-const G_E_MOp_Def_DRES      E_G)
(declare-const G_E_MOp_Def_RRES      E_G)
(declare-const G_E_MOp_Def_DSUB      E_G)
(declare-const G_E_MOp_Def_RSUB      E_G)
;; TExp
(declare-const G_E_TExp_Def_SExp     E_G)
(declare-const G_E_TExp_Def_FExp     E_G)
;; FExp
(declare-const G_E_FExpId            E_G)
(declare-const G_E_FExpNum           E_G)
(declare-const G_E_FExpUMinus        E_G)
(declare-const G_E_FExpPar           E_G)
(declare-const G_E_FExpDot_Id        E_G)
(declare-const G_E_FExpDot_PropId    E_G)
(declare-const G_E_FExpBinExp1       E_G)
(declare-const G_E_FExpBinExp2       E_G)
(declare-const G_E_FExpBinOp         E_G)
(declare-const G_E_FExp_Def_Id       E_G)
(declare-const G_E_FExp_Def_Num      E_G)
(declare-const G_E_FExp_Def_UMinus   E_G)
(declare-const G_E_FExp_Def_Par      E_G)
(declare-const G_E_FExp_Def_Dot      E_G)
(declare-const G_E_FExp_Def_Bin      E_G)
;; FEOp
(declare-const G_E_FEOp_Def_Plus     E_G)
(declare-const G_E_FEOp_Def_Minus    E_G)
(declare-const G_E_FEOp_Def_Times    E_G)
(declare-const G_E_FEOp_Def_Div      E_G)

```



```

;; SExp
(declare-const G_E_SExpTD E_G)
(declare-const G_E_SExpSDef E_G)
(declare-const G_E_SExpCard E_G)
(declare-const G_E_SExp_Def_TD E_G)
(declare-const G_E_SExp_Def_SetDef E_G)
(declare-const G_E_SExp_Def_Empty E_G)
(declare-const G_E_SExp_Def_Card E_G)
;; IDef
(declare-const G_E_IDef_SExt_SEs E_G)
(declare-const G_E_IDef_CntSet_TD E_G)
(declare-const G_E_IDef_CntSet_PEs E_G)
(declare-const G_E_IDef_Def_SExt E_G)
(declare-const G_E_IDef_Def_CntSet E_G)
;; IExp
(declare-const G_E_IExp_Def_IDef E_G)
(declare-const G_E_IExp_Def_SDs E_G)
(declare-const G_E_IExpSDs_SDef E_G)
;; SDef
(declare-const G_E_SDef_IExp E_G)
(declare-const G_E_SDef_SOp E_G)
;; SOp
(declare-const G_E_SOp_Def_Domain E_G)
(declare-const G_E_SOp_Def_Range E_G)
(declare-const G_E_SOp_Def_Union E_G)
(declare-const G_E_SOp_Def_Intersection E_G)
(declare-const G_E_SOp_Def_CrossProduct E_G)
(declare-const G_E_SOp_Def_SetMinus E_G)
(declare-const G_E_SOp_Def_RelComp E_G)
(declare-const G_E_SOp_Def_None E_G)
;; Special 'Null' constant to check totality
(declare-const G_E_Null E_G)

(assert (distinct
  MM_Null
  MM_Num
  MM_Name
  MM_TypeDesignator
  MM_TypeDesignatorNat
  MM_TypeDesignatorInt
  MM_TypeDesignatorId
  MM_FreeExpression
  MM_FreeExpId
  MM_FreeExpNum
  MM_FreeExpUMinus
  MM_FreeExpPar
  MM_FreeExpDot
  MM_FreeExpBin
  MM_FreeExpBinOp

```

```

MM_FreeExpBinOp_Plus
MM_FreeExpBinOp_Minus
MM_FreeExpBinOp_Times
MM_FreeExpBinOp_Div
MM_SetExpression
MM_SetExpressionId
MM_SetExpressionDef
MM_SetExpressionEmpty
MM_PropEdge
MM_PropEdgePred
MM_PropEdgeMod
MM_EdgeOperatorUn
MM_EdgeOperatorBin
MM_EdgeOperatorMod
MM_UOpCard
MM_UOpThe
MM_UOpNone
MM_BOpEQ
MM_BOpNEQ
MM_BOpIN
MM_BOpLT
MM_BOpLEQ
MM_BOpGT
MM_BOpGEQ
MM_BOpSubsetEQ
MM_MOpDRES
MM_MOpRRES
MM_MOpDSUB
MM_MOpRSUB
MM_Assertion
MM_VCLObj
MM_Pair
MM_SetElement
MM_InsideDef
MM_SetExtension
MM_ConstrainedSet
MM_SetInsideExpression
MM_InsideExpSDs
MM_SetDef
MM_SetDefOp
MM_SOp_Domain
MM_SOp_Range
MM_SOp_Union
MM_SOp_Intersection
MM_SOp_CrossProduct
MM_SOp_SetMinus
MM_SOp_RelComp
MM_SOp_None))

```

```

(assert (distinct
  G_Null
  G_Id
  G_PE
  G_PEP
  G_PEM
  G_BEOp
  G_UEOp
  G_MOp
  G_UEOp_Card
  G_UEOp_The
  G_UEOp_None
  G_BEOp_EQ
  G_BEOp_NEQ
  G_BEOp_IN
  G_BEOp_LT
  G_BEOp_LEQ
  G_BEOp_GT
  G_BEOp_GEQ
  G_BEOp_SubsetEQ
  G_MOp_DRES
  G_MOp_RRES
  G_MOp_DSUB
  G_MOp_RSUB
  G_TD
  G_TD_Nat
  G_TD_Int
  G_TD_Id
  G_FExp
  G_FExpId
  G_FExpNum
  G_FExpUMinus
  G_FExpPar
  G_FExpDot
  G_FExpBin
  G_FEOp
  G_FEOp_Plus
  G_FEOp_Minus
  G_FEOp_Times
  G_FEOp_Div
  G_SExp
  G_SExpTD
  G_SExpSDef
  G_SExpEmpty
  G_A
  G_0
  G_P
  G_SE
  G_IDef

```

```

G_IDef_SExt
G_IDef_CntSet
G_IExp
G_IExp_SDs
G_IExp_IDef
G_SDef
G_SOp
G_SOp_Domain
G_SOp_Range
G_SOp_Union
G_SOp_Intersection
G_SOp_CrossProduct
G_SOp_SetMinus
G_SOp_RelComp
G_SOp_None))

(assert (distinct
  MM_ENull
  MM_ETypeDesignatorId
  MM_EITypeDesignatorId
  MM_EITypeDesignatorNat
  MM_EITypeDesignatorInt
  MM_EISetExpression
  MM_EIFreeExp
  MM_EFreeExpNum
  MM_EIFreeExpNum
  MM_EFreeExpId
  MM_EIFreeExpId
  MM_EFreeExpUMinus
  MM_EIFreeExpUMinus
  MM_EFreeExpPar
  MM_EIFreeExpPar
  MM_EFreeExpDotId
  MM_EFreeExpDotPropId
  MM_EIFreeExpDot
  MM_EFreeExpBinExp1
  MM_EFreeExpBinExp2
  MM_EFreeExpBinOp
  MM_EIFreeExpBinExp
  MM_EIFreeExpBinOp_Plus
  MM_EIFreeExpBinOp_Minus
  MM_EIFreeExpBinOp_Times
  MM_EIFreeExpBinOp_Div
  MM_ESetExpressionId
  MM_EISetExpressionId
  MM_ESetExpressionCard
  MM_EISetExpressionCard
  MM_ESetExpressionDef
  MM_EISetExpressionDef

```

```

MM_EISetExpressionEmpty
MM_EPropEdgeTarget
MM_EPropEdgePredBOp
MM_EPropEdgePredName
MM_EPropEdgeModMOp
MM_EIPropEdgeMod
MM_EIPropEdgePred
MM_EPropEdgePredUOp
MM_EIUOp_Card
MM_EIUOp_The
MM_EIUOp_None
MM_EIBOp_EQ
MM_EIBOp_NEQ
MM_EIBOp_In
MM_EIBOp_LT
MM_EIBOp_LEQ
MM_EIBOp_GT
MM_EIBOp_GEQ
MM_EIMOp_DRES
MM_EIMOp_RRES
MM_EIMOp_DSUB
MM_EIMOp_RSUB
MM_EIBOp_SubsetEQ
MM_EAssertion_Id
MM_EVCLObj_Id
MM_EPair_Id1
MM_EPair_Id2
MM_EIVCLObj
MM_EIPair
MM_ESetExtension_Elems
MM_EConstrainedSet_Desig
MM_EConstrainedSet_PropEdge
MM_EISetExtension
MM_EIConstrainedSet
MM_EIInsideDef
MM_EIInsideExpSDs
MM_ESetDef_insideExp
MM_ESetDef_sdop
MM_EInsideExpSDs_setDefs
MM_EISOp_Domain
MM_EISOp_Range
MM_EISOp_Union
MM_EISOp_Intersection
MM_EISOp_CrossProduct
MM_EISOp_SetMinus
MM_EISOp_RelComp
MM_EISOp_None))

(assert (distinct

```

G_E_Null
 G_E_TD_Id
 G_E_TD_Def_Id
 G_E_TD_Def_Nat
 G_E_TD_Def_Int
 G_E_TExp_Def_SExp
 G_E_TExp_Def_FExp
 G_E_FExpId
 G_E_FExp_Def_Id
 G_E_FExpNum
 G_E_FExp_Def_Num
 G_E_FExpUMinus
 G_E_FExp_Def_UMinus
 G_E_FExpPar
 G_E_FExp_Def_Par
 G_E_FExpBinExp1
 G_E_FExpBinExp2
 G_E_FExpBinOp
 G_E_FExp_Def_Bin
 G_E_FExpDot_Id
 G_E_FExpDot_PropId
 G_E_FExp_Def_Dot
 G_E_FEOp_Def_Plus
 G_E_FEOp_Def_Minus
 G_E_FEOp_Def_Times
 G_E_FEOp_Def_Div
 G_E_SExpTD
 G_E_SExp_Def_TD
 G_E_SExpSDef
 G_E_SExp_Def_SetDef
 G_E_SExpCard
 G_E_SExp_Def_Card
 G_E_SExp_Def_Empty
 G_E_PE_TExp
 G_E_PE_PEP
 G_E_PE_PEM
 G_E_PEP_UEOp
 G_E_PEP_Id
 G_E_PEP_BEOp
 G_E_PEM_MOp
 G_E_UEOp_Def_Card
 G_E_UEOp_Def_The
 G_E_UEOp_Def_None
 G_E_BEOp_Def_Eq
 G_E_BEOp_Def_Neq
 G_E_BEOp_Def_In
 G_E_BEOp_Def_LT
 G_E_BEOp_Def_LEQ
 G_E_BEOp_Def_GT

```

G_E_BEOp_Def_GEQ
G_E_BEOp_Def_SUBSETEQ
G_E_MOp_Def_DRES
G_E_MOp_Def_RRES
G_E_MOp_Def_DSUB
G_E_MOp_Def_RSUB
G_E_A_Id
G_E_SE_Def_0
G_E_SE_Def_P
G_E_O_Id
G_E_P_Id_1
G_E_P_Id_2
G_E_IDef_SExt_SEs
G_E_IDef_CntSet_TD
G_E_IDef_CntSet_PEs
G_E_IDef_Def_SExt
G_E_IDef_Def_CntSet
G_E_IExp_Def_IDef
G_E_IExp_Def_SDs
G_E_SDef_IExp
G_E_SDef_SOp
G_E_IExpSDs_SDef
G_E_SOp_Def_Domain
G_E_SOp_Def_Range
G_E_SOp_Def_Union
G_E_SOp_Def_Intersection
G_E_SOp_Def_CrossProduct
G_E_SOp_Def_SetMinus
G_E_SOp_Def_RelComp
G_E_SOp_Def_None))

```

```

(define-fun Map_V ((v V_MM)) V_G
  (ite (= v MM_Num)          G_Num
    (ite (= v MM_Name)       G_Id
      (ite (= v MM_TypeDesignator) G_TD
        (ite (= v MM_TypeDesignatorNat) G_TD_Nat
          (ite (= v MM_TypeDesignatorInt) G_TD_Int
            (ite (= v MM_TypeDesignatorId) G_TD_Id
              (ite (= v MM_FreeExpression) G_FExp
                (ite (= v MM_FreeExpId) G_FExpId
                  (ite (= v MM_FreeExpNum) G_FExpNum
                    (ite (= v MM_FreeExpUMinus) G_FExpUMinus
                      (ite (= v MM_FreeExpPar) G_FExpPar
                        (ite (= v MM_FreeExpDot) G_FExpDot
                          (ite (= v MM_FreeExpBin) G_FExpBin
                            (ite (= v MM_FreeExpBinOp) G_FEOp
                              (ite (= v MM_FreeExpBinOp_Plus) G_FEOp_Plus
                                (ite (= v MM_FreeExpBinOp_Minus) G_FEOp_Minus
                                  (ite (= v MM_FreeExpBinOp_Times) G_FEOp_Times

```

```

(ite (= v MM_FreeExpBinOp_Div)      G_FEOp_Div
(ite (= v MM_TypeDesignator)        G_TD
(ite (= v MM_SetExpression)          G_SExp
(ite (= v MM_SetExpressionId)        G_SExpTD
(ite (= v MM_SetExpressionDef)       G_SExpSDef
(ite (= v MM_SetExpressionEmpty)     G_SExpEmpty
(ite (= v MM_PropEdge)               G_PE
(ite (= v MM_PropEdgePred)           G_PEP
(ite (= v MM_PropEdgeMod)            G_PEM
(ite (= v MM_EdgeOperatorUn)         G_UEOp
(ite (= v MM_EdgeOperatorBin)        G_BEOp
(ite (= v MM_EdgeOperatorMod)        G_MOp
(ite (= v MM_UOpCard)                G_UEOp_Card
(ite (= v MM_UOpThe)                 G_UEOp_The
(ite (= v MM_UOpNone)                G_UEOp_None
(ite (= v MM_BOpEQ)                  G_BEOp_EQ
(ite (= v MM_BOpNEQ)                 G_BEOp_NEQ
(ite (= v MM_BOpIN)                  G_BEOp_IN
(ite (= v MM_BOpLT)                  G_BEOp_LT
(ite (= v MM_BOpLEQ)                 G_BEOp_LEQ
(ite (= v MM_BOpGT)                  G_BEOp_GT
(ite (= v MM_BOpGEQ)                 G_BEOp_GEQ
(ite (= v MM_BOpSubsetEQ)            G_BEOp_SubsetEQ
(ite (= v MM_MOpDRES)                 G_MOp_DRES
(ite (= v MM_MOpRRES)                 G_MOp_RRES
(ite (= v MM_MOpDSUB)                 G_MOp_DSUB
(ite (= v MM_MOpRSUB)                 G_MOp_RSUB
(ite (= v MM_PropEdge)               G_PE
(ite (= v MM_Assertion)               G_A
(ite (= v MM_VCLObj)                 G_O
(ite (= v MM_Pair)                   G_P
(ite (= v MM_SetElement)              G_SE
(ite (= v MM_InsideDef)               G_IDef
(ite (= v MM_SetExtension)            G_IDef_SExt
(ite (= v MM_ConstrainedSet)          G_IDef_CntSet
(ite (= v MM_SetInsideExpression)     G_IExp
(ite (= v MM_InsideExpSDs)            G_IExp_SDs
(ite (= v MM_SetDef)                  G_SDef
(ite (= v MM_SetDefOp)                G_SOp
(ite (= v MM_SOp_Domain)              G_SOp_Domain
(ite (= v MM_SOp_Range)               G_SOp_Range
(ite (= v MM_SOp_Union)               G_SOp_Union
(ite (= v MM_SOp_Intersection)        G_SOp_Intersection
(ite (= v MM_SOp_CrossProduct)        G_SOp_CrossProduct
(ite (= v MM_SOp_SetMinus)            G_SOp_SetMinus
(ite (= v MM_SOp_RelComp)             G_SOp_RelComp
(ite (= v MM_SOp_None)                G_SOp_None
G_Null))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))

```



```

(define-fun Map_E ((e E_MM)) E_G
  (ite (= e MM_ETypeDesignatorId)      G_E_TD_Id
    (ite (= e MM_EITypeDesignatorId)    G_E_TD_Def_Id
      (ite (= e MM_EITypeDesignatorNat) G_E_TD_Def_Nat
        (ite (= e MM_EITypeDesignatorInt) G_E_TD_Def_Int
          (ite (= e MM_EFreeExpNum)      G_E_FExpNum
            (ite (= e MM_EIFreeExpNum)    G_E_FExp_Def_Num
              (ite (= e MM_EFreeExpId)    G_E_FExpId
                (ite (= e MM_EIFreeExpId)  G_E_FExp_Def_Id
                  (ite (= e MM_EFreeExpUMinus) G_E_FExpUMinus
                    (ite (= e MM_EIFreeExpUMinus) G_E_FExp_Def_UMinus
                      (ite (= e MM_EFreeExpPar) G_E_FExpPar
                        (ite (= e MM_EIFreeExpPar) G_E_FExp_Def_Par
                          (ite (= e MM_EFreeExpDotId) G_E_FExpDot_Id
                            (ite (= e MM_EFreeExpDotPropId) G_E_FExpDot_PropId
                              (ite (= e MM_EIFreeExpDot) G_E_FExp_Def_Dot
                                (ite (= e MM_EFreeExpBinExp1) G_E_FExpBinExp1
                                  (ite (= e MM_EFreeExpBinExp2) G_E_FExpBinExp2
                                    (ite (= e MM_EFreeExpBinOp) G_E_FExpBinOp
                                      (ite (= e MM_EIFreeExpBinExp) G_E_FExp_Def_Bin
                                        (ite (= e MM_EIFreeExpBinOp_Plus) G_E_FEOp_Def_Plus
                                          (ite (= e MM_EIFreeExpBinOp_Minus) G_E_FEOp_Def_Minus
                                            (ite (= e MM_EIFreeExpBinOp_Times) G_E_FEOp_Def_Times
                                              (ite (= e MM_EIFreeExpBinOp_Div) G_E_FEOp_Def_Div
                                                (ite (= e MM_ESetExpressionCard) G_E_SExpCard
                                                  (ite (= e MM_EISetExpressionCard) G_E_SExp_Def_Card
                                                    (ite (= e MM_ESetExpressionId) G_E_SExpTD
                                                      (ite (= e MM_EISetExpressionId) G_E_SExp_Def_TD
                                                        (ite (= e MM_EISetExpressionEmpty) G_E_SExp_Def_Empty
                                                          (ite (= e MM_ESetExpressionDef) G_E_SExpSDef
                                                            (ite (= e MM_EISetExpressionDef) G_E_SExp_Def_SetDef
                                                              (ite (= e MM_EISetExpression) G_E_TExp_Def_SExp
                                                                (ite (= e MM_EIFreeExp) G_E_TExp_Def_FExp
                                                                  (ite (= e MM_EAssertion_Id) G_E_A_Id
                                                                    (ite (= e MM_EPropEdgeTarget) G_E_PE_TExp
                                                                      (ite (= e MM_EPropEdgePredBOp) G_E_PEP_BEOp
                                                                        (ite (= e MM_EPropEdgePredName) G_E_PEP_Id
                                                                          (ite (= e MM_EPropEdgeModMOp) G_E_PEM_MOp
                                                                            (ite (= e MM_EPropEdgePredUOp) G_E_PEP_UOp
                                                                              (ite (= e MM_EIPropEdgePred) G_E_PE_PEP
                                                                                (ite (= e MM_EIPropEdgeMod) G_E_PE_PEM
                                                                                  (ite (= e MM_EIUOp_Card) G_E_UOp_Def_Card
                                                                                    (ite (= e MM_EIUOp_The) G_E_UOp_Def_The
                                                                                      (ite (= e MM_EIUOp_None) G_E_UOp_Def_None
                                                                                        (ite (= e MM_EIBOp_EQ) G_E_BEOp_Def_Eq
                                                                                          (ite (= e MM_EIBOp_NEQ) G_E_BEOp_Def_Neq
                                                                                            (ite (= e MM_EIBOp_In) G_E_BEOp_Def_In
                                                                                              (ite (= e MM_EIBOp_LT) G_E_BEOp_Def_LT
                                                                                                (ite (= e MM_EIBOp_LEQ) G_E_BEOp_Def_LEQ

```

```

(ite (= e MM_EIBOp_GT) G_E_BEOp_Def_GT
(ite (= e MM_EIBOp_GEQ) G_E_BEOp_Def_GEQ
(ite (= e MM_EIBOp_SubsetEQ) G_E_BEOp_Def_SUBSETEQ
(ite (= e MM_EIMOp_DRES) G_E_MOp_Def_DRES
(ite (= e MM_EIMOp_RRES) G_E_MOp_Def_RRES
(ite (= e MM_EIMOp_DSUB) G_E_MOp_Def_DSUB
(ite (= e MM_EIMOp_RSUB) G_E_MOp_Def_RSUB
(ite (= e MM_EIVCLObj) G_E_SE_Def_0
(ite (= e MM_EIPair) G_E_SE_Def_P
(ite (= e MM_EVCLObj_Id) G_E_0_Id
(ite (= e MM_EPair_Id1) G_E_P_Id_1
(ite (= e MM_EPair_Id2) G_E_P_Id_2
(ite (= e MM_ESetExtension_Elems) G_E_IDef_SExt_SEs
(ite (= e MM_EConstrainedSet_Desig) G_E_IDef_CntSet_TD
(ite (= e MM_EConstrainedSet_PropEdge) G_E_IDef_CntSet_PEs
(ite (= e MM_EISetExtension) G_E_IDef_Def_SExt
(ite (= e MM_EIConstrainedSet) G_E_IDef_Def_CntSet
(ite (= e MM_EIInsideDef) G_E_IExp_Def_IDef
(ite (= e MM_EIInsideExpSDs) G_E_IExp_Def_SDs
(ite (= e MM_ESetDef_insideExp) G_E_SDef_IExp
(ite (= e MM_ESetDef_sdop) G_E_SDef_SOOp
(ite (= e MM_EIInsideExpSDs_setDefs) G_E_IExpSDs_SDef
(ite (= e MM_EISOp_Domain) G_E_SOOp_Def_Domain
(ite (= e MM_EISOp_Range) G_E_SOOp_Def_Range
(ite (= e MM_EISOp_Union) G_E_SOOp_Def_Union
(ite (= e MM_EISOp_Intersection) G_E_SOOp_Def_Intersection
(ite (= e MM_EISOp_CrossProduct) G_E_SOOp_Def_CrossProduct
(ite (= e MM_EISOp_SetMinus) G_E_SOOp_Def_SetMinus
(ite (= e MM_EISOp_RelComp) G_E_SOOp_Def_RelComp
(ite (= e MM_EISOp_None) G_E_SOOp_Def_None
G_E_Null))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))

(push)
(echo "Testing function 'Map_V' (1) --> sat")
(assert (= (Map_V MM_PropEdge) G_PE))
(check-sat)
(pop)

(push)
(echo "Testing function 'Map_V' (2) --> sat")
(assert (= (Map_V MM_VCLObj) G_0))
(check-sat)
(pop)

(push)
(echo "Testing function 'Map_V' (3) --> unsat")
(assert (= (Map_V MM_SetElement) G_P))
(check-sat)
(pop)

```

```

(push)
(echo "Checking Totality of 'Map_V' --> sat")
(assert (forall ((vmm V_MM))
  (=> (= (Map_V vmm) G_Null) (= vmm MM_Null))))
(check-sat)
(pop)

(push)
(echo "Checking injectiveness of 'Map_V' --> sat")
(assert (forall ((vmm1 V_MM) (vmm2 V_MM))
  (=> (= (Map_V vmm1) (Map_V vmm2)) (= vmm1 vmm2))))
(check-sat)
(pop)

(push)
(echo "Checking Surjectiveness of 'Map_V' --> sat")
(assert (forall ((vg V_G))
  (exists ((vmm V_MM))
    (= (Map_V vmm) vg))))
(check-sat)
(pop)

; (push)
; (echo "Checking Surjectiveness of 'Map_V' (2)--> sat")
; (declare-fun svmm (V_G) V_MM)
; (assert (forall ((vg V_G))
;   (= (Map_V (svmm vg)) vg)))
; (check-sat)
; (pop)

(push)
(echo "Testing function 'Map_E' (1) --> sat")
(assert (= (Map_E MM_EAssertion_Id) G_E_A_Id))
(check-sat)
(pop)

(push)
(echo "Testing function 'Map_E' (2) --> sat")
(assert (= (Map_E MM_ESetExtension_Elems) G_E_IDef_SExt_SEs))
(check-sat)
(pop)

(push)
(echo "Testing function 'Map_E' (3) --> unsat")
(assert (= (Map_E MM_EAssertion_Id) G_E_SOp_Def_None))
(check-sat)
(pop)

```

```

(push)
(echo "Checking Totality of 'Map_E' --> sat")
(assert (forall ((emm E_MM))
  (=> (= (Map_E emm) G_E_Null) (= emm MM_ENull))))
(check-sat)
(pop)

(push)
(echo "Checking injectiveness of 'Map_E' --> sat")
(assert (forall ((emm1 E_MM) (emm2 E_MM))
  (=> (= (Map_E emm1) (Map_E emm2)) (= emm1 emm2))))
(check-sat)
(pop)

(push)
(echo "Checking Surjectiveness of 'Map_E' --> sat")
(assert (forall ((eg E_G))
  (exists ((emm E_MM))
    (= (Map_E emm) eg))))
(check-sat)
(pop)

; (push)
; (echo "Checking surjectiveness of 'Map_E' (2)--> sat")
; (declare-fun semm (E_G) E_MM)
; (assert (forall ((eg E_G))
;   (= (Map_E (semm eg)) eg)))
; (check-sat)
; (pop)

(define-fun Target_MM ((e E_MM)) V_MM
  (ite (= e MM_ETypeDesignatorId) MM_Name
    (ite (= e MM_EITypeDesignatorId) MM_TypeDesignator
      (ite (= e MM_EITypeDesignatorNat) MM_TypeDesignator
        (ite (= e MM_EITypeDesignatorInt) MM_TypeDesignator
          (ite (= e MM_EISetExpression) MM_Expression
            (ite (= e MM_EIFreeExp) MM_Expression
              (ite (= e MM_EFreeExpNum) MM_Num
                (ite (= e MM_EIFreeExpNum) MM_FreeExpression
                  (ite (= e MM_EFreeExpId) MM_Name
                    (ite (= e MM_EIFreeExpId) MM_FreeExpression
                      (ite (= e MM_EFreeExpUMinus) MM_FreeExpression
                        (ite (= e MM_EIFreeExpUMinus) MM_FreeExpression
                          (ite (= e MM_EFreeExpPar) MM_FreeExpression
                            (ite (= e MM_EIFreeExpPar) MM_FreeExpression
                              (ite (= e MM_EFreeExpDotId) MM_Name
                                (ite (= e MM_EFreeExpDotPropId) MM_Name
                                  (ite (= e MM_EIFreeExpDot) MM_FreeExpression

```

(ite (= e MM_EFreeExpBinExp1)	MM_FreeExpression
(ite (= e MM_EFreeExpBinExp2)	MM_FreeExpression
(ite (= e MM_EFreeExpBinOp)	MM_FreeExpBinOp
(ite (= e MM_EIFreeExpBinExp)	MM_FreeExpression
(ite (= e MM_EIFreeExpBinOp_Plus)	MM_FreeExpBinOp
(ite (= e MM_EIFreeExpBinOp_Minus)	MM_FreeExpBinOp
(ite (= e MM_EIFreeExpBinOp_Times)	MM_FreeExpBinOp
(ite (= e MM_EIFreeExpBinOp_Div)	MM_FreeExpBinOp
(ite (= e MM_ESetExpressionId)	MM_TypeDesignator
(ite (= e MM_ESetExpressionDef)	MM_SetDef
(ite (= e MM_ESetExpressionCard)	MM_SetExpression
(ite (= e MM_EISetExpressionId)	MM_SetExpression
(ite (= e MM_EISetExpressionDef)	MM_SetExpression
(ite (= e MM_EISetExpressionCard)	MM_SetExpression
(ite (= e MM_EISetExpressionEmpty)	MM_SetExpression
(ite (= e MM_EPropEdgeTarget)	MM_Expression
(ite (= e MM_EPropEdgePredBOp)	MM_EdgeOperatorBin
(ite (= e MM_EPropEdgePredName)	MM_Name
(ite (= e MM_EPropEdgeModMOp)	MM_EdgeOperatorMod
(ite (= e MM_EIPropEdgeMod)	MM_PropEdge
(ite (= e MM_EIPropEdgePred)	MM_PropEdge
(ite (= e MM_EPropEdgePredUOp)	MM_EdgeOperatorUn
(ite (= e MM_EIUOp_Card)	MM_EdgeOperatorUn
(ite (= e MM_EIUOp_The)	MM_EdgeOperatorUn
(ite (= e MM_EIUOp_None)	MM_EdgeOperatorUn
(ite (= e MM_EIBOp_EQ)	MM_EdgeOperatorBin
(ite (= e MM_EIBOp_NEQ)	MM_EdgeOperatorBin
(ite (= e MM_EIBOp_In)	MM_EdgeOperatorBin
(ite (= e MM_EIBOp_LT)	MM_EdgeOperatorBin
(ite (= e MM_EIBOp_LEQ)	MM_EdgeOperatorBin
(ite (= e MM_EIBOp_GT)	MM_EdgeOperatorBin
(ite (= e MM_EIBOp_GEQ)	MM_EdgeOperatorBin
(ite (= e MM_EIBOp_SubsetEQ)	MM_EdgeOperatorBin
(ite (= e MM_EIMOp_DRES)	MM_EdgeOperatorMod
(ite (= e MM_EIMOp_RRES)	MM_EdgeOperatorMod
(ite (= e MM_EIMOp_DSUB)	MM_EdgeOperatorMod
(ite (= e MM_EIMOp_RSUB)	MM_EdgeOperatorMod
(ite (= e MM_EAssertion_Id)	MM_Name
(ite (= e MM_EIVCLObj)	MM_SetElement
(ite (= e MM_EIPair)	MM_SetElement
(ite (= e MM_EVCLObj_Id)	MM_Name
(ite (= e MM_EPair_Id1)	MM_Name
(ite (= e MM_EPair_Id2)	MM_Name
(ite (= e MM_ESetExtension_Elems)	MM_SetElement
(ite (= e MM_EConstrainedSet_Design)	MM_TypeDesignator
(ite (= e MM_EConstrainedSet_PropEdge)	MM_PropEdge
(ite (= e MM_EISetExtension)	MM_InsideDef
(ite (= e MM_EIConstrainedSet)	MM_InsideDef
(ite (= e MM_EIInsideDef)	MM_SetInsideExpression

```

(ite (= e MM_EIInsideExpSDs) MM_SetInsideExpression
(ite (= e MM_ESetDef_insideExp) MM_SetInsideExpression
(ite (= e MM_ESetDef_sdop) MM_SetDefOp
(ite (= e MM_EIInsideExpSDs_setDefs) MM_SetDef
(ite (= e MM_EISOp_Domain) MM_SetDefOp
(ite (= e MM_EISOp_Range) MM_SetDefOp
(ite (= e MM_EISOp_Union) MM_SetDefOp
(ite (= e MM_EISOp_Intersection) MM_SetDefOp
(ite (= e MM_EISOp_CrossProduct) MM_SetDefOp
(ite (= e MM_EISOp_SetMinus) MM_SetDefOp
(ite (= e MM_EISOp_RelComp) MM_SetDefOp
(ite (= e MM_EISOp_None) MM_SetDefOp
MM_Null))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))

(define-fun Source_MM ((e E_MM)) V_MM
  (ite (= e MM_ETypeDesignatorId) MM_TypeDesignatorId
  (ite (= e MM_EITypeDesignatorId) MM_TypeDesignatorId
  (ite (= e MM_EITypeDesignatorNat) MM_TypeDesignatorNat
  (ite (= e MM_EITypeDesignatorInt) MM_TypeDesignatorInt
  (ite (= e MM_EISetExpression) MM_SetExpression
  (ite (= e MM_EIFreeExp) MM_FreeExpression
  (ite (= e MM_EFreeExpNum) MM_FreeExpNum
  (ite (= e MM_EIFreeExpNum) MM_FreeExpNum
  (ite (= e MM_EFreeExpId) MM_FreeExpId
  (ite (= e MM_EIFreeExpId) MM_FreeExpId
  (ite (= e MM_EFreeExpUMinus) MM_FreeExpUMinus
  (ite (= e MM_EIFreeExpUMinus) MM_FreeExpUMinus
  (ite (= e MM_EFreeExpPar) MM_FreeExpPar
  (ite (= e MM_EIFreeExpPar) MM_FreeExpPar
  (ite (= e MM_EFreeExpDotId) MM_FreeExpDot
  (ite (= e MM_EFreeExpDotPropId) MM_FreeExpDot
  (ite (= e MM_EIFreeExpDot) MM_FreeExpDot
  (ite (= e MM_EFreeExpBinExp1) MM_FreeExpBin
  (ite (= e MM_EFreeExpBinExp2) MM_FreeExpBin
  (ite (= e MM_EFreeExpBinOp) MM_FreeExpBin
  (ite (= e MM_EIFreeExpBinExp) MM_FreeExpBin
  (ite (= e MM_EIFreeExpBinOp_Plus) MM_FreeExpBinOp_Plus
  (ite (= e MM_EIFreeExpBinOp_Minus) MM_FreeExpBinOp_Minus
  (ite (= e MM_EIFreeExpBinOp_Times) MM_FreeExpBinOp_Times
  (ite (= e MM_EIFreeExpBinOp_Div) MM_FreeExpBinOp_Div
  (ite (= e MM_ESetExpressionId) MM_SetExpressionId
  (ite (= e MM_ESetExpressionDef) MM_SetExpressionDef
  (ite (= e MM_ESetExpressionCard) MM_SetExpressionCard
  (ite (= e MM_EISetExpressionId) MM_SetExpressionCard
  (ite (= e MM_EISetExpressionDef) MM_SetExpressionDef
  (ite (= e MM_EISetExpressionCard) MM_SetExpressionCard
  (ite (= e MM_EISetExpressionEmpty) MM_SetExpressionEmpty
  (ite (= e MM_EPropEdgeTarget) MM_PropEdge
  (ite (= e MM_EPropEdgePredB0p) MM_PropEdgePred

```

```

(ite (= e MM_EPropEdgePredName)      MM_PropEdgePred
(ite (= e MM_EPropEdgeModMOp)        MM_PropEdgeMod
(ite (= e MM_EIPropEdgeMod)          MM_PropEdgeMod
(ite (= e MM_EIPropEdgePred)         MM_PropEdgePred
(ite (= e MM_EPropEdgePredUOp)       MM_PropEdgePred
(ite (= e MM_EIUOp_Card)              MM_UOpCard
(ite (= e MM_EIUOp_The)               MM_UOpThe
(ite (= e MM_EIUOp_None)              MM_UOpNone
(ite (= e MM_EIBOp_EQ)                MM_BOpEQ
(ite (= e MM_EIBOp_NEQ)               MM_BOpNEQ
(ite (= e MM_EIBOp_In)                MM_BOpIN
(ite (= e MM_EIBOp_LT)                MM_BOpLT
(ite (= e MM_EIBOp_LEQ)               MM_BOpLEQ
(ite (= e MM_EIBOp_GT)                MM_BOpGT
(ite (= e MM_EIBOp_GEQ)               MM_BOpGEQ
(ite (= e MM_EIBOp_SubsetEQ)          MM_BOpSubsetEQ
(ite (= e MM_EIMOp_DRES)              MM_MOpDRES
(ite (= e MM_EIMOp_RRES)              MM_MOpRRES
(ite (= e MM_EIMOp_DSUB)              MM_MOpDSUB
(ite (= e MM_EIMOp_RSUB)              MM_MOpRSUB
(ite (= e MM_EAssertion_Id)           MM_Assertion
(ite (= e MM_EIVCLObj)                MM_VCLObj
(ite (= e MM_EIPair)                  MM_Pair
(ite (= e MM_EVCLObj_Id)              MM_VCLObj
(ite (= e MM_EPair_Id1)               MM_Pair
(ite (= e MM_EPair_Id2)               MM_Pair
(ite (= e MM_ESetExtension_Elems)     MM_SetExtension
(ite (= e MM_EConstrainedSet_Desig)   MM_ConstrainedSet
(ite (= e MM_EConstrainedSet_PropEdge) MM_ConstrainedSet
(ite (= e MM_EISetExtension)          MM_SetExtension
(ite (= e MM_EIConstrainedSet)        MM_ConstrainedSet
(ite (= e MM_EIInsideDef)             MM_InsideDef
(ite (= e MM_EIInsideExpSDs)          MM_InsideExpSDs
(ite (= e MM_ESetDef_insideExp)       MM_SetDef
(ite (= e MM_ESetDef_sdop)            MM_SetDef
(ite (= e MM_EInsideExpSDs_setDefs)   MM_InsideExpSDs
(ite (= e MM_EISOp_Domain)            MM_SOp_Domain
(ite (= e MM_EISOp_Range)             MM_SOp_Range
(ite (= e MM_EISOp_Union)             MM_SOp_Union
(ite (= e MM_EISOp_Intersection)      MM_SOp_Intersection
(ite (= e MM_EISOp_CrossProduct)      MM_SOp_CrossProduct
(ite (= e MM_EISOp_SetMinus)          MM_SOp_SetMinus
(ite (= e MM_EISOp_RelComp)           MM_SOp_RelComp
(ite (= e MM_EISOp_None)              MM_SOp_None
MM_Null))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))
(define-fun Target_G ((e E_G)) V_G
  (ite (= e G_E_TD_Id)      G_Id
    (ite (= e G E TD Def Id) G_TD

```

(ite (= e G_E_TD_Def_Nat)	G_TD
(ite (= e G_E_TD_Def_Int)	G_TD
(ite (= e G_E_TExp_Def_SExp)	G_TExp
(ite (= e G_E_TExp_Def_FExp)	G_TExp
(ite (= e G_E_FExpNum)	G_Num
(ite (= e G_E_FExpId)	G_Id
(ite (= e G_E_FExpUMinus)	G_FExp
(ite (= e G_E_FExpPar)	G_FExp
(ite (= e G_E_FExpBinExp1)	G_FExp
(ite (= e G_E_FExpBinExp2)	G_FExp
(ite (= e G_E_FExpBinOp)	G_FEOp
(ite (= e G_E_FExp_Def_Id)	G_FExp
(ite (= e G_E_FExp_Def_Num)	G_FExp
(ite (= e G_E_FExp_Def_UMinus)	G_FExp
(ite (= e G_E_FExp_Def_Par)	G_FExp
(ite (= e G_E_FExp_Def_Bin)	G_FExp
(ite (= e G_E_FExpDot_Id)	G_Id
(ite (= e G_E_FExpDot_PropId)	G_Id
(ite (= e G_E_FExp_Def_Dot)	G_FExp
(ite (= e G_E_FEOp_Def_Plus)	G_FEOp
(ite (= e G_E_FEOp_Def_Minus)	G_FEOp
(ite (= e G_E_FEOp_Def_Times)	G_FEOp
(ite (= e G_E_FEOp_Def_Div)	G_FEOp
(ite (= e G_E_SExpTD)	G_TD
(ite (= e G_E_SExpSDef)	G_SDef
(ite (= e G_E_SExpCard)	G_SExp
(ite (= e G_E_SExp_Def_TD)	G_SExp
(ite (= e G_E_SExp_Def_SetDef)	G_SExp
(ite (= e G_E_SExp_Def_Card)	G_SExp
(ite (= e G_E_SExp_Def_Empty)	G_SExp
(ite (= e G_E_PE_TExp)	G_TExp
(ite (= e G_E_PE_PEP)	G_PE
(ite (= e G_E_PE_PEM)	G_PE
(ite (= e G_E_PEP_UEOp)	G_UEOp
(ite (= e G_E_PEP_Id)	G_Id
(ite (= e G_E_PEP_BEOp)	G_BEOp
(ite (= e G_E_PEM_MOp)	G_MOp
(ite (= e G_E_UEOp_Def_Card)	G_UEOp
(ite (= e G_E_UEOp_Def_The)	G_UEOp
(ite (= e G_E_UEOp_Def_None)	G_UEOp
(ite (= e G_E_BEOp_Def_Eq)	G_BEOp
(ite (= e G_E_BEOp_Def_Neq)	G_BEOp
(ite (= e G_E_BEOp_Def_In)	G_BEOp
(ite (= e G_E_BEOp_Def_LT)	G_BEOp
(ite (= e G_E_BEOp_Def_LEQ)	G_BEOp
(ite (= e G_E_BEOp_Def_GT)	G_BEOp
(ite (= e G_E_BEOp_Def_GEQ)	G_BEOp
(ite (= e G_E_BEOp_Def_SUBSETEQ)	G_BEOp
(ite (= e G_E_MOp_Def_DRES)	G_MOp


```
(ite (= e G_E_MOp_Def_RRES) G_MOp
(ite (= e G_E_MOp_Def_DSUB) G_MOp
(ite (= e G_E_MOp_Def_RSUB) G_MOp
(ite (= e G_E_A_Id) G_Id
(ite (= e G_E_0_Id) G_Id
(ite (= e G_E_P_Id_1) G_Id
(ite (= e G_E_P_Id_2) G_Id
(ite (= e G_E_SE_Def_0) G_SE
(ite (= e G_E_SE_Def_P) G_SE
(ite (= e G_E_IDef_SExt_SEs) G_SE
(ite (= e G_E_IDef_CntSet_TD) G_TD
(ite (= e G_E_IDef_CntSet_PEs) G_PE
(ite (= e G_E_IDef_Def_SExt) G_IDef
(ite (= e G_E_IDef_Def_CntSet) G_IDef
(ite (= e G_E_IExp_Def_IDef) G_IExp
(ite (= e G_E_IExp_Def_SDs) G_IExp
(ite (= e G_E_SDef_IExp) G_IExp
(ite (= e G_E_SDef_SO p) G_SO p
(ite (= e G_E_IExpSDs_SDef) G_SDef
(ite (= e G_E_SO p_Def_Domain) G_SO p
(ite (= e G_E_SO p_Def_Range) G_SO p
(ite (= e G_E_SO p_Def_Union) G_SO p
(ite (= e G_E_SO p_Def_Intersection) G_SO p
(ite (= e G_E_SO p_Def_CrossProduct) G_SO p
(ite (= e G_E_SO p_Def_SetMinus) G_SO p
(ite (= e G_E_SO p_Def_RelComp) G_SO p
(ite (= e G_E_SO p_Def_None) G_SO p
G_Null)))))))))
(define-fun Source_G ((e E_G)) V_G
(ite (= e G_E_TD_Id) G_TD_Id
(ite (= e G_E_TD_Def_Id) G_TD_Id
(ite (= e G_E_TD_Def_Nat) G_TD_Nat
(ite (= e G_E_TD_Def_Int) G_TD_Int
(ite (= e G_E_TExp_Def_SExp) G_SExp
(ite (= e G_E_TExp_Def_FExp) G_FExp
(ite (= e G_E_FExpNum) G_FExpNum
(ite (= e G_E_FExpId) G_FExpId
(ite (= e G_E_FExpUMinus) G_FExpUMinus
(ite (= e G_E_FExpPar) G_FExpPar
(ite (= e G_E_FExpBinExp1) G_FExpBin
(ite (= e G_E_FExpBinExp2) G_FExpBin
(ite (= e G_E_FExpBinOp) G_FExpBin
(ite (= e G_E_FExp_Def_Id) G_FExpId
(ite (= e G_E_FExp_Def_Num) G_FExpNum
(ite (= e G_E_FExp_Def_UMinus) G_FExpUMinus
(ite (= e G_E_FExp_Def_Par) G_FExpPar
(ite (= e G_E_FExp_Def_Bin) G_FExpBin
(ite (= e G_E_FExpDot Id) G_FExpDot
```

(ite (= e G_E_FExpDot_PropId)	G_FExpDot
(ite (= e G_E_FExp_Def_Dot)	G_FExpDot
(ite (= e G_E_FEOp_Def_Plus)	G_FEOp_Plus
(ite (= e G_E_FEOp_Def_Minus)	G_FEOp_Minus
(ite (= e G_E_FEOp_Def_Times)	G_FEOp_Times
(ite (= e G_E_FEOp_Def_Div)	G_FEOp_Div
(ite (= e G_E_SExpTD)	G_SExpTD
(ite (= e G_E_SExpSDef)	G_SExpSDef
(ite (= e G_E_SExpCard)	G_SExpCard
(ite (= e G_E_SExp_Def_TD)	G_SExpTD
(ite (= e G_E_SExp_Def_SetDef)	G_SExpSDef
(ite (= e G_E_SExp_Def_Card)	G_SExpCard
(ite (= e G_E_SExp_Def_Empty)	G_SExpEmpty
(ite (= e G_E_PE_TExp)	G_PE
(ite (= e G_E_PE_PEP)	G_PEP
(ite (= e G_E_PE_PEM)	G_PEM
(ite (= e G_E_PEP_UEOp)	G_PEP
(ite (= e G_E_PEP_Id)	G_PEP
(ite (= e G_E_PEP_BEOp)	G_PEP
(ite (= e G_E_PEM_MOOp)	G_PEM
(ite (= e G_E_UEOp_Def_Card)	G_UEOp_Card
(ite (= e G_E_UEOp_Def_The)	G_UEOp_The
(ite (= e G_E_UEOp_Def_None)	G_UEOp_None
(ite (= e G_E_BEOp_Def_Eq)	G_BEOp_EQ
(ite (= e G_E_BEOp_Def_Neq)	G_BEOp_NEQ
(ite (= e G_E_BEOp_Def_In)	G_BEOp_IN
(ite (= e G_E_BEOp_Def_LT)	G_BEOp_LT
(ite (= e G_E_BEOp_Def_LEQ)	G_BEOp_LEQ
(ite (= e G_E_BEOp_Def_GT)	G_BEOp_GT
(ite (= e G_E_BEOp_Def_GEQ)	G_BEOp_GEQ
(ite (= e G_E_BEOp_Def_SUBSETEQ)	G_BEOp_SubsetEQ
(ite (= e G_E_MOOp_Def_DRES)	G_MOOp_DRES
(ite (= e G_E_MOOp_Def_RRES)	G_MOOp_RRES
(ite (= e G_E_MOOp_Def_DSUB)	G_MOOp_DSUB
(ite (= e G_E_MOOp_Def_RSUB)	G_MOOp_RSUB
(ite (= e G_E_A_Id)	G_A
(ite (= e G_E_O_Id)	G_O
(ite (= e G_E_P_Id_1)	G_P
(ite (= e G_E_P_Id_2)	G_P
(ite (= e G_E_SE_Def_O)	G_O
(ite (= e G_E_SE_Def_P)	G_P
(ite (= e G_E_IDef_SEExt_SEs)	G_IDef_SEExt
(ite (= e G_E_IDef_CntSet_TD)	G_IDef_CntSet
(ite (= e G_E_IDef_CntSet_PEs)	G_IDef_CntSet
(ite (= e G_E_IDef_Def_SEExt)	G_IDef_SEExt
(ite (= e G_E_IDef_Def_CntSet)	G_IDef_CntSet
(ite (= e G_E_IExp_Def_IDef)	G_IDef
(ite (= e G_E_IExp_Def_SDs)	G_IExp_SDs
(ite (= e G_E_SDef_IExp)	G_SDef

```

(ite (= e G_E_SDef_SOp)          G_SDef
(ite (= e G_E_IExpSDs_SDef)      G_IExp_SDs
(ite (= e G_E_SOp_Def_Domain)    G_SOp_Domain
(ite (= e G_E_SOp_Def_Range)     G_SOp_Range
(ite (= e G_E_SOp_Def_Union)     G_SOp_Union
(ite (= e G_E_SOp_Def_Intersection) G_SOp_Intersection
(ite (= e G_E_SOp_Def_CrossProduct) G_SOp_CrossProduct
(ite (= e G_E_SOp_Def_SetMinus)   G_SOp_SetMinus
(ite (= e G_E_SOp_Def_RelComp)    G_SOp_RelComp
(ite (= e G_E_SOp_Def_None)       G_SOp_None
G_Null))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))

(push)
(echo "Testing the 'Target_MM' function (1) --> sat")
(assert (= (Target_MM MM_EAssertion_Id) MM_Name))
(check-sat)
(pop)

(push)
(echo "Testing the target 'Target_MM' function (2) --> sat")
(assert (= (Target_MM MM_EISOp_Range) MM_SetDefOp))
(check-sat)
(pop)

(push)
(echo "Testing the 'Target_MM' function (3) --> unsat")
(assert (= (Target_MM MM_EISetExtension) MM_PropEdge))
(check-sat)
(pop)

(push)
(echo "Checking totality of 'Target_MM' --> sat")
(assert (forall ((emm E_MM))
  (=> (= (Target_MM emm) MM_Null) (= emm MM_ENull))))
(check-sat)
(pop)

(push)
(echo "Testing the 'Source_MM' function (1) --> sat")
(assert (= (Source_MM MM_EAssertion_Id) MM_Assertion))
(check-sat)
(pop)

(push)
(echo "Testing the 'Source_MM' function (2) --> sat")
(assert (= (Source_MM MM_EISOp_Range) MM_SOp_Range))
(check-sat)
(pop)

```

```

(push)
(echo "Testing the 'Source_MM' function (3) --> unsat")
(assert (= (Source_MM MM_EISetExtension) MM_SetDef))
(check-sat)
(pop)

(push)
(echo "Checking Totality of 'Source_MM' --> sat")
(assert (forall ((emm E_MM))
  (=> (= (Source_MM emm) MM_Null) (= emm MM_ENull))))
(check-sat)
(pop)

(push)
(echo "Checking totality of 'Target_G' ->sat")
(assert (forall ((eg E_G))
  (=> (= (Target_G eg) G_Null) (= eg G_E_Null))))
(check-sat)
(pop)

(push)
(echo "Checking that the target function 'Target_MM' is preserved -> sat")
(assert (forall ((emm1 E_MM))
  (= (Map_V (Target_MM emm1)) (Target_G (Map_E emm1)))))
(check-sat)
(pop)

(push)
(echo "Testing the 'Source_G' function (1)-> sat")
(assert (= (Source_G G_E_IDef_Def_SExt) G_IDef_SExt))
(check-sat)
(pop)

(push)
(echo "Testing the 'Source_G' function (2) -> sat")
(assert (= (Source_G G_E_SDef_SOp) G_SDef))
(check-sat)
(pop)

(push)
(echo "Testing the 'Source_G' function (3) -> unsat")
(assert (= (Source_G G_E_IDef_CntSet_TD) G_TD))
(check-sat)
(pop)

(push)
(echo "Checking Totality of 'Source_G' ->sat")
(assert (forall ((eg E_G))
  (=> (= (Source_G eg) G_Null) (= eg G_E_Null))))

```

```

(check-sat)
(pop)

(push)
(echo "Checking that the source function 'Source_MM' is preserved -> sat")
(assert (forall ((emm1 E_MM))
  (= (Map_V (Source_MM emm1)) (Source_G (Map_E emm1)))))
(check-sat)
(pop)

```

C.1.2 Z3 Proof Output

```

Testing function 'Map_V' (1) --> sat
sat
Testing function 'Map_V' (2) --> sat
sat
Testing function 'Map_V' (3) --> unsat
unsat
Checking Totality of 'Map_V' --> sat
sat
Checking injectiveness of 'Map_V' --> sat
sat
Checking Surjectiveness of 'Map_V' --> sat
sat
Testing function 'Map_E' (1) --> sat
sat
Testing function 'Map_E' (2) --> sat
sat
Testing function 'Map_E' (3) --> unsat
unsat
Checking Totality of 'Map_E' --> sat
sat
Checking injectiveness of 'Map_E' --> sat
sat
Checking Surjectiveness of 'Map_E' --> sat
sat
Testing the 'Target_MM' function (1) --> sat
sat
Testing the target 'Target_MM' function (2) --> sat
sat
Testing the 'Target_MM' function (3) --> unsat
unsat
Checking totality of 'Target_MM' --> sat
sat
Testing the 'Source_MM' function (1) --> sat
sat
Testing the 'Source_MM' function (2) --> sat
sat
Testing the 'Source_MM' function (3) --> unsat

```

```

unsat
Checking Totality of 'Source_MM' --> sat
sat
Checking totality of 'Target_G' ->sat
sat
Checking that the target function 'Target_MM' is preserved -> sat
sat
Testing the 'Source_G' function (1)-> sat
sat
Testing the 'Source_G' function (2) -> sat
sat
Testing the 'Source_G' function (3) -> unsat
unsat
Checking Totality of 'Source_G' ->sat
sat
Checking that the source function 'Source_MM' is preserved -> sat
sat

```

C.2 Structural diagrams

```

(set-option :mbqi true)
(set-option :macro-finder true)
(set-option :pull-nested-quantifiers true)
(set-option :produce-unsat-cores true)
(set-option :produce-models true)

(declare-sort V_MM)
(declare-sort E_MM)

(declare-sort V_G)
(declare-sort E_G)

(declare-const MM_Name          V_MM)
(declare-const MM_Num           V_MM)
(declare-const MM_Bool          V_MM)
(declare-const MM_Assertion     V_MM)
(declare-const MM_TypeDesignator V_MM)
(declare-const MM_SetDef        V_MM)
;; 'Mult'
(declare-const MM_Mult          V_MM)
(declare-const MM_MSeq          V_MM)
(declare-const MM_MOne          V_MM)
(declare-const MM_MOpt          V_MM)
(declare-const MM_MMany         V_MM)
(declare-const MM_MRange        V_MM)
(declare-const MM_MOneToMany    V_MM)
(declare-const MM_UBound        V_MM)
(declare-const MM_UBoundNum     V_MM)

```

```

(declare-const MM_UBoundStar      V_MM)
;; 'SetKind'
(declare-const MM_SetKind          V_MM)
(declare-const MM_SetKind_Value    V_MM)
(declare-const MM_SetKind_Class    V_MM)
;; 'SDElem'
(declare-const MM_SDElem           V_MM)
;; 'Constant'
(declare-const MM_Constant          V_MM)
;; 'Relation Edge'
(declare-const MM_RelEdge           V_MM)
;; 'PropEdgeDef'
(declare-const MM_PropEdgeDef       V_MM)
;; 'Set'
(declare-const MM_Set               V_MM)
(declare-const MM_PrimarySet        V_MM)
(declare-const MM_DerivedSet        V_MM)
;; 'SetDefObject'
(declare-const MM_SetDefObject      V_MM)
;; 'SDiag'
(declare-const MM_SDiag             V_MM)
;; Special 'Null' constant to check totality
(declare-const MM_Null              V_MM)

;; Mult
(declare-const MM_E_I_MSeq          E_MM)
(declare-const MM_E_I_MOne          E_MM)
(declare-const MM_E_I_MOpt          E_MM)
(declare-const MM_E_I_MMAny         E_MM)
(declare-const MM_E_I_MOneToMany    E_MM)
(declare-const MM_E_I_MRange        E_MM)
(declare-const MM_E_MRange_lb       E_MM)
(declare-const MM_E_MRange_ub       E_MM)
(declare-const MM_E_I_UBoundNum     E_MM)
(declare-const MM_E_I_UBoundStar    E_MM)
;; 'SetKind'
(declare-const MM_E_I_SetKind_Value E_MM)
(declare-const MM_E_I_SetKind_Class E_MM)
;; 'Constant'
(declare-const MM_E_I_Constant       E_MM)
(declare-const MM_E_Constant_name    E_MM)
(declare-const MM_E_Constant_TD      E_MM)
;; 'PropEdgeDef'
(declare-const MM_E_PropEdgeDef_mult E_MM)
(declare-const MM_E_PropEdgeDef_tgt  E_MM)
(declare-const MM_E_PropEdgeDef_id   E_MM)
;; 'RelEdge'
(declare-const MM_E_I_RelEdge        E_MM)
(declare-const MM_E_RelEdge_name     E_MM)

```

```

(declare-const MM_E_RelEdge_Src      E_MM)
(declare-const MM_E_RelEdge_MultS    E_MM)
(declare-const MM_E_RelEdge_Tgt      E_MM)
(declare-const MM_E_RelEdge_MultT    E_MM)
;; 'Set'
(declare-const MM_E_I_Set            E_MM)
(declare-const MM_E_I_PrimarySet     E_MM)
(declare-const MM_E_PrimarySet_name  E_MM)
(declare-const MM_E_PrimarySet_isDef E_MM)
(declare-const MM_E_PrimarySet_lcs   E_MM)
(declare-const MM_E_PrimarySet_lis   E_MM)
(declare-const MM_E_PrimarySet_hio   E_MM)
(declare-const MM_E_PrimarySet_his   E_MM)
(declare-const MM_E_PrimarySet_kind  E_MM)
(declare-const MM_E_PrimarySet_lps   E_MM)
(declare-const MM_E_I_DerivedSet     E_MM)
(declare-const MM_E_DerivedSet_name  E_MM)
(declare-const MM_E_DerivedSet_def   E_MM)
;; 'SetDefObject'
(declare-const MM_E_SetDefObject_objName E_MM)
;; 'Sdiag'
(declare-const MM_E_Sdiag_elements   E_MM)
(declare-const MM_E_Sdiag_invariants E_MM)
;; Special 'Null' constant to check totality
(declare-const MM_ENull              E_MM)

(declare-const G_Num                V_G)
(declare-const G_Id                 V_G)
(declare-const G_Bool               V_G)
(declare-const G_A                  V_G)
(declare-const G_O                  V_G)
(declare-const G_TD                 V_G)
(declare-const G_SDef               V_G)
;; M
(declare-const G_M                  V_G)
(declare-const G_M_One              V_G)
(declare-const G_M_Opt              V_G)
(declare-const G_M_Some             V_G)
(declare-const G_M_Many             V_G)
(declare-const G_M_Seq              V_G)
(declare-const G_M_Range            V_G)
(declare-const G_UBound             V_G)
(declare-const G_UBound_Num         V_G)
(declare-const G_UBound_Star        V_G)
;; 'SK'
(declare-const G_SK                 V_G)
(declare-const G_SK_Value           V_G)
(declare-const G_SK_Class           V_G)
;; 'SDE'

```



```

(declare-const G_SDE          V_G)
;; 'C'
(declare-const G_C            V_G)
;; 'RE'
(declare-const G_RE           V_G)
;; 'PED'
(declare-const G_PED           V_G)
;; 'Set'
(declare-const G_Set           V_G)
(declare-const G_DSet          V_G)
(declare-const G_PSet          V_G)
;; 'SD'
(declare-const G_SD            V_G)
;; Special 'Null' constant to check totality
(declare-const G_Null          V_G)

;; 0
(declare-const G_E_O_Id        E_G)
;; Mult
(declare-const G_E_M_Def_opt    E_G)
(declare-const G_E_M_Def_one    E_G)
(declare-const G_E_M_Def_some   E_G)
(declare-const G_E_M_Def_many   E_G)
(declare-const G_E_M_Def_seq    E_G)
(declare-const G_E_M_Def_range  E_G)
(declare-const G_E_MRange_lb    E_G)
(declare-const G_E_MRange_ub    E_G)
(declare-const G_E_UBound_Def_Num E_G)
(declare-const G_E_UBound_Def_Star E_G)
;; 'SK'
(declare-const G_E_SK_Def_Value E_G)
(declare-const G_E_SK_Def_Class E_G)
;; 'SDE'
(declare-const G_E_SDE_Def_C     E_G)
(declare-const G_E_SDE_Def_RE    E_G)
(declare-const G_E_SDE_Def_Set   E_G)
;; 'C'
(declare-const G_E_C_TD          E_G)
(declare-const G_E_C_Id         E_G)
;; 'RE'
(declare-const G_E_RE_Id         E_G)
(declare-const G_E_RE_Src_TD     E_G)
(declare-const G_E_RE_Src_M      E_G)
(declare-const G_E_RE_Tgt_TD     E_G)
(declare-const G_E_RE_Tgt_M      E_G)
;; 'PED'
(declare-const G_E_PED_M         E_G)
(declare-const G_E_PED_TD        E_G)

```

```

(declare-const G_E_PED_Id          E_G)
;; 'Set'
(declare-const G_E_Set_Def_PSet    E_G)
(declare-const G_E_Set_Def_DSet    E_G)
(declare-const G_E_PSet_Id         E_G)
(declare-const G_E_PSet_SK         E_G)
(declare-const G_E_PSet_isDef      E_G)
(declare-const G_E_PSet-Cs         E_G)
(declare-const G_E_PSet_PEDs       E_G)
(declare-const G_E_PSet_As         E_G)
(declare-const G_E_PSet_hiOs       E_G)
(declare-const G_E_PSet_hiPSs      E_G)
(declare-const G_E_DSet_SDef       E_G)
(declare-const G_E_DSet_Id         E_G)
;; 'SD'
(declare-const G_E_SD_SDEs         E_G)
(declare-const G_E_SD_As           E_G)
;; Special 'Null' constant to check totality
(declare-const G_E_Null            E_G)

(assert (distinct
  MM_Null
  MM_Num
  MM_Name
  MM_Bool
  MM_TypeDesignator
  MM_Assertion
  MM_SetDef
  MM_Mult
  MM_MSeq
  MM_MOne
  MM_MOpt
  MM_MMany
  MM_MRange
  MM_MOneToMany
  MM_UBound
  MM_UBoundNum
  MM_UBoundStar
  MM_SetKind
  MM_SetKind_Class
  MM_SetKind_Value
  MM_SDElem
  MM_Constant
  MM_RelEdge
  MM_PropEdgeDef
  MM_Set
  MM_PrimarySet
  MM_DerivedSet
  MM_SetDefObject

```

```

MM_SDiag))

(assert (distinct
  G_Null
  G_Id
  G_Num
  G_Bool
  G_A
  G_O
  G_TD
  G_SDef
  G_M
  G_M_One
  G_M_Opt
  G_M_Some
  G_M_Many
  G_M_Seq
  G_M_Range
  G_UBound
  G_UBound_Num
  G_UBound_Star
  G_SK
  G_SK_Value
  G_SK_Class
  G_SDE
  G_C
  G_RE
  G_PED
  G_Set
  G_DSet
  G_PSet
  G_SD))

(assert (distinct
  MM_ENull
  MM_E_I_MSeq
  MM_E_I_MOne
  MM_E_I_MOpt
  MM_E_I_MMany
  MM_E_I_MOneToMany
  MM_E_I_MRange
  MM_E_MRange_lb
  MM_E_MRange_ub
  MM_E_I_UBoundNum
  MM_E_I_UBoundStar
  MM_E_I_SetKind_Value
  MM_E_I_SetKind_Class
  MM_E_I_Constant
  MM_E_Constant_TD

```

```

MM_E_Constant_name
MM_E_I_RelEdge
MM_E_RelEdge_name
MM_E_RelEdge_Src
MM_E_RelEdge_Tgt
MM_E_RelEdge_MultS
MM_E_RelEdge_MultT
MM_E_PropEdgeDef_mult
MM_E_PropEdgeDef_tgt
MM_E_PropEdgeDef_id
MM_E_I_Set
MM_E_I_PrimarySet
MM_E_PrimarySet_name
MM_E_PrimarySet_isDef
MM_E_PrimarySet_lcs
MM_E_PrimarySet_lis
MM_E_PrimarySet_hio
MM_E_PrimarySet_his
MM_E_PrimarySet_kind
MM_E_PrimarySet_lps
MM_E_I_DerivedSet
MM_E_DerivedSet_name
MM_E_DerivedSet_def
MM_E_SetDefObject_objName
MM_E_SDiag_elements
MM_E_SDiag_invariants))

(assert (distinct
  G_E_Null
  G_E_O_Id
  G_E_M_Def_opt
  G_E_M_Def_one
  G_E_M_Def_some
  G_E_M_Def_many
  G_E_M_Def_seq
  G_E_M_Def_range
  G_E_MRange_lb
  G_E_MRange_ub
  G_E_UBound_Def_Num
  G_E_UBound_Def_Star
  G_E_SK_Def_Value
  G_E_SK_Def_Class
  G_E_SDE_Def_C
  G_E_C_Id
  G_E_C_TD
  G_E_SDE_Def_RE
  G_E_RE_Id
  G_E_RE_Src_TD
  G_E_RE_Tgt_TD

```

```

G_E_RE_Src_M
G_E_RE_Tgt_M
G_E_PED_M
G_E_PED_TD
G_E_PED_Id
G_E_SDE_Def_Set
G_E_Set_Def_PSet
G_E_Set_Def_DSet
G_E_PSet_Id
G_E_PSet_SK
G_E_PSet_isDef
G_E_PSet-Cs
G_E_PSet_PEDs
G_E_PSet_As
G_E_PSet_hiOs
G_E_PSet_hiPSs
G_E_DSet_Id
G_E_DSet_SDef
G_E_SD_SDEs
G_E_SD_As))

(define-fun Map_V ((v V_MM)) V_G
  (ite (= v MM_Num)          G_Num
    (ite (= v MM_Name)       G_Id
      (ite (= v MM_Bool)     G_Bool
        (ite (= v MM_TypeDesignator) G_TD
          (ite (= v MM_Assertion) G_A
            (ite (= v MM_SetDef) G_SDef
              (ite (= v MM_Mult) G_M
                (ite (= v MM_MSeq) G_M_Seq
                  (ite (= v MM_MOne) G_M_One
                    (ite (= v MM_MOpt) G_M_Opt
                      (ite (= v MM_MMAny) G_M_Many
                        (ite (= v MM_MRange) G_M_Range
                          (ite (= v MM_MOneToMany) G_M_Some
                            (ite (= v MM_UBound) G_UBound
                              (ite (= v MM_UBoundNum) G_UBound_Num
                                (ite (= v MM_UBoundStar) G_UBound_Star
                                  (ite (= v MM_SetKind) G_SK
                                    (ite (= v MM_SetKind_Value) G_SK_Value
                                      (ite (= v MM_SetKind_Class) G_SK_Class
                                        (ite (= v MM_SDElem) G_SDE
                                          (ite (= v MM_Constant) G_C
                                            (ite (= v MM_RelEdge) G_RE
                                              (ite (= v MM_PropEdgeDef) G_PED
                                                (ite (= v MM_Set) G_Set
                                                  (ite (= v MM_PrimarySet) G_PSet
                                                    (ite (= v MM_DerivedSet) G_DSet
                                                      (ite (= v MM_SetDefObject) G_0

```

```

(ite (= v MM_SDiag) G_SD
G_Null))))))))))))))))))))))))))))))

(define-fun Map_E ((e E_MM)) E_G
  (ite (= e MM_E_I_MSeq) G_E_M_Def_seq
    (ite (= e MM_E_I_MOne) G_E_M_Def_one
      (ite (= e MM_E_I_MOpt) G_E_M_Def_opt
        (ite (= e MM_E_I_MMany) G_E_M_Def_many
          (ite (= e MM_E_I_MOneToMany) G_E_M_Def_some
            (ite (= e MM_E_I_MRange) G_E_M_Def_range
              (ite (= e MM_E_MRange_lb) G_E_MRange_lb
                (ite (= e MM_E_MRange_ub) G_E_MRange_ub
                  (ite (= e MM_E_I_UBoundNum) G_E_UBound_Def_Num
                    (ite (= e MM_E_I_UBoundStar) G_E_UBound_Def_Star
                      (ite (= e MM_E_I_SetKind_Value) G_E_SK_Def_Value
                        (ite (= e MM_E_I_SetKind_Class) G_E_SK_Def_Class
                          (ite (= e MM_E_I_Constant) G_E_SDE_Def_C
                            (ite (= e MM_E_Constant_name) G_E_C_Id
                              (ite (= e MM_E_Constant_TD) G_E_C_TD
                                (ite (= e MM_E_I_RelEdge) G_E_SDE_Def_RE
                                  (ite (= e MM_E_RelEdge_name) G_E_RE_Id
                                    (ite (= e MM_E_RelEdge_Src) G_E_RE_Src_TD
                                      (ite (= e MM_E_RelEdge_Tgt) G_E_RE_Tgt_TD
                                        (ite (= e MM_E_RelEdge_MultS) G_E_RE_Src_M
                                          (ite (= e MM_E_RelEdge_MultT) G_E_RE_Tgt_M
                                            (ite (= e MM_E_PropEdgeDef_mult) G_E_PED_M
                                              (ite (= e MM_E_PropEdgeDef_tgt) G_E_PED_TD
                                                (ite (= e MM_E_PropEdgeDef_id) G_E_PED_Id
                                                  (ite (= e MM_E_I_Set) G_E_SDE_Def_Set
                                                    (ite (= e MM_E_I_PrimarySet) G_E_Set_Def_PSet
                                                      (ite (= e MM_E_PrimarySet_name) G_E_PSet_Id
                                                        (ite (= e MM_E_PrimarySet_isDef) G_E_PSet_isDef
                                                          (ite (= e MM_E_PrimarySet_lcs) G_E_PSet-Cs
                                                            (ite (= e MM_E_PrimarySet_lis) G_E_PSet_As
                                                              (ite (= e MM_E_PrimarySet_hio) G_E_PSet_hiOs
                                                                (ite (= e MM_E_PrimarySet_his) G_E_PSet_hiPSs
                                                                  (ite (= e MM_E_PrimarySet_kind) G_E_PSet_SK
                                                                    (ite (= e MM_E_PrimarySet_lps) G_E_PSet_PEDs
                                                                      (ite (= e MM_E_I_DerivedSet) G_E_Set_Def_DSet
                                                                        (ite (= e MM_E_DerivedSet_name) G_E_DSet_Id
                                                                          (ite (= e MM_E_DerivedSet_def) G_E_DSet_SDef
                                                                            (ite (= e MM_E_SetDefObject_objName) G_E_O_Id
                                                                              (ite (= e MM_E_SDiag_elements) G_E_SD_SDEs
                                                                                (ite (= e MM_E_SDiag_invariants) G_E_SD_As
                                                                                  G_E_Null))))))))))))))))))))))))))))))))))

(push)
(echo "Testing function 'Map_V' (1) --> sat")
(assert (= (Map_V MM_SDElem) G_SDE))

```

```

(check-sat)
(pop)

(push)
(echo "Testing function 'Map_V' (2) --> sat")
(assert (= (Map_V MM_PropEdgeDef) G_PED))
(check-sat)
(pop)

(push)
(echo "Testing function 'Map_V' (3) --> unsat")
(assert (= (Map_V MM_SetKind_Class) G_SK_Value))
(check-sat)
(pop)

(push)
(echo "Checking Totality of 'Map_V' --> sat")
(assert (forall ((vmm V_MM))
  (=> (= (Map_V vmm) G_Null) (= vmm MM_Null))))
(check-sat)
(pop)

(push)
(echo "Checking injectiveness of 'Map_V' --> sat")
(assert (forall ((vmm1 V_MM) (vmm2 V_MM))
  (=> (= (Map_V vmm1) (Map_V vmm2)) (= vmm1 vmm2))))
(check-sat)
(pop)

(push)
(echo "Checking Surjectiveness of 'Map_V' (1) --> sat")
(assert (forall ((vg V_G))
  (exists ((vmm V_MM))
    (= (Map_V vmm) vg))))
(check-sat)
(pop)

; (push)
; (echo "Checking Surjectiveness of 'Map_V' (2)->sat")
; (declare-fun svmm (V_G) V_MM)
; (assert (forall ((vg V_G))
;   (= (Map_V (svmm vg)) vg)))
; (check-sat)
; (pop)

(push)
(echo "Testing function 'Map_E' (1) --> sat")
(assert (= (Map_E MM_E_I_Constant) G_E_SDE_Def_C))
(check-sat)

```

```

(pop)

(push)
(echo "Testing function 'Map_E' (2) --> sat")
(assert (= (Map_E MM_E_I_RelEdge) G_E_SDE_Def_RE))
(check-sat)
(pop)

(push)
(echo "Testing function 'Map_E' (3) --> unsat")
(assert (= (Map_E MM_E_MRange_lb) G_E_M_Def_opt))
(check-sat)
(pop)

(push)
(echo "Checking Totality of 'Map_E' --> sat")
(assert (forall ((emm E_MM))
  (=> (= (Map_E emm) G_E_Null) (= emm MM_ENull))))
(check-sat)
(pop)

(push)
(echo "Checking injectiveness of 'Map_E' --> sat")
(assert (forall ((emm1 E_MM) (emm2 E_MM))
  (=> (= (Map_E emm1) (Map_E emm2)) (= emm1 emm2))))
(check-sat)
(pop)

(push)
(echo "Checking Surjectiveness of 'Map_E' (1) --> sat")
(assert (forall ((eg E_G))
  (exists ((emm E_MM))
    (= (Map_E emm) eg))))
(check-sat)
(pop)

; (push)
; (echo "Checking surjectiveness of 'Map_E' (2) -> sat")
; (declare-fun semm (E_G) E_MM)
; (assert (forall ((eg E_G))
;   (= (Map_E (semm eg)) eg)))
; (check-sat)
; (pop)

(define-fun Target_MM ((e E_MM)) V_MM
  (ite (= e MM_E_I_MSeq) MM_Mult
    (ite (= e MM_E_I_MOne) MM_Mult
      (ite (= e MM_E_I_MOpt) MM_Mult
        (ite (= e MM_E_I_MMany) MM_Mult

```



```

(ite (= e MM_E_I_MOneToMany) MM_Mult
(ite (= e MM_E_I_MRange) MM_Mult
(ite (= e MM_E_MRange_lb) MM_Num
(ite (= e MM_E_MRange_ub) MM_UBound
(ite (= e MM_E_I_UBoundNum) MM_UBound
(ite (= e MM_E_I_UBoundStar) MM_UBound
(ite (= e MM_E_I_SetKind_Value) MM_SetKind
(ite (= e MM_E_I_SetKind_Class) MM_SetKind
(ite (= e MM_E_I_Constant) MM_SDElem
(ite (= e MM_E_Constant_name) MM_Name
(ite (= e MM_E_Constant_TD) MM_TypeDesignator
(ite (= e MM_E_I_RelEdge) MM_SDElem
(ite (= e MM_E_RelEdge_name) MM_Name
(ite (= e MM_E_RelEdge_Src) MM_TypeDesignator
(ite (= e MM_E_RelEdge_Tgt) MM_TypeDesignator
(ite (= e MM_E_RelEdge_MultS) MM_Mult
(ite (= e MM_E_RelEdge_MultT) MM_Mult
(ite (= e MM_E_PropEdgeDef_mult) MM_Mult
(ite (= e MM_E_PropEdgeDef_tgt) MM_TypeDesignator
(ite (= e MM_E_PropEdgeDef_id) MM_Name
(ite (= e MM_E_I_Set) MM_SDElem
(ite (= e MM_E_I_PrimarySet) MM_Set
(ite (= e MM_E_PrimarySet_name) MM_Name
(ite (= e MM_E_PrimarySet_isDef) MM_Bool
(ite (= e MM_E_PrimarySet_lcs) MM_Constant
(ite (= e MM_E_PrimarySet_lis) MM_Assertion
(ite (= e MM_E_PrimarySet_hio) MM_SetDefObject
(ite (= e MM_E_PrimarySet_his) MM_PrimarySet
(ite (= e MM_E_PrimarySet_kind) MM_SetKind
(ite (= e MM_E_PrimarySet_lps) MM_PropEdgeDef
(ite (= e MM_E_I_DerivedSet) MM_Set
(ite (= e MM_E_DerivedSet_name) MM_Name
(ite (= e MM_E_DerivedSet_def) MM_SetDef
(ite (= e MM_E_SetDefObject_objName) MM_Name
(ite (= e MM_E_SDiag_elements) MM_SDElem
(ite (= e MM_E_SDiag_invariants) MM_Assertion
MM_Null))))))))))))))))))))))))))))))))))))))

```

```

(define-fun Source_MM ((e E_MM)) V_MM
  (ite (= e MM_E_I_MSeq) MM_MSeq
    (ite (= e MM_E_I_MOne) MM_MOne
      (ite (= e MM_E_I_MOpt) MM_MOpt
        (ite (= e MM_E_I_MMany) MM_MMany
          (ite (= e MM_E_I_MOneToMany) MM_MOneToMany
            (ite (= e MM_E_I_MRange) MM_MRange
              (ite (= e MM_E_MRange_lb) MM_MRange
                (ite (= e MM_E_MRange_ub) MM_MRange
                  (ite (= e MM_E_I_UBoundNum) MM_UBoundNum
                    (ite (= e MM_E_I_UBoundStar) MM_UBoundStar

```

```

(ite (= e MM_E_I_SetKind_Value) MM_SetKind_Value
(ite (= e MM_E_I_SetKind_Class) MM_SetKind_Class
(ite (= e MM_E_I_Constant) MM_Constant
(ite (= e MM_E_Constant_name) MM_Constant
(ite (= e MM_E_Constant_TD) MM_Constant
(ite (= e MM_E_I_RelEdge) MM_RelEdge
(ite (= e MM_E_RelEdge_name) MM_RelEdge
(ite (= e MM_E_RelEdge_Src) MM_RelEdge
(ite (= e MM_E_RelEdge_Tgt) MM_RelEdge
(ite (= e MM_E_RelEdge_MultS) MM_RelEdge
(ite (= e MM_E_RelEdge_MultT) MM_RelEdge
(ite (= e MM_E_PropEdgeDef_mult) MM_PropEdgeDef
(ite (= e MM_E_PropEdgeDef_tgt) MM_PropEdgeDef
(ite (= e MM_E_PropEdgeDef_id) MM_PropEdgeDef
(ite (= e MM_E_I_Set) MM_Set
(ite (= e MM_E_I_PrimarySet) MM_PrimarySet
(ite (= e MM_E_PrimarySet_name) MM_PrimarySet
(ite (= e MM_E_PrimarySet_isDef) MM_PrimarySet
(ite (= e MM_E_PrimarySet_lcs) MM_PrimarySet
(ite (= e MM_E_PrimarySet_lis) MM_PrimarySet
(ite (= e MM_E_PrimarySet_hio) MM_PrimarySet
(ite (= e MM_E_PrimarySet_his) MM_PrimarySet
(ite (= e MM_E_PrimarySet_kind) MM_PrimarySet
(ite (= e MM_E_PrimarySet_lps) MM_PrimarySet
(ite (= e MM_E_I_DerivedSet) MM_DerivedSet
(ite (= e MM_E_DerivedSet_name) MM_DerivedSet
(ite (= e MM_E_DerivedSet_def) MM_DerivedSet
(ite (= e MM_E_SetDefObject_objName) MM_SetDefObject
(ite (= e MM_E_SDiag_elements) MM_SDiag
(ite (= e MM_E_SDiag_invariants) MM_SDiag
MM_Null))))))))))))))))))))))))))))))))))))))

```

```

(define-fun Target_G ((e E_G)) V_G
  (ite (= e G_E_O_Id) G_Id
    (ite (= e G_E_M_Def_opt) G_M
      (ite (= e G_E_M_Def_one) G_M
        (ite (= e G_E_M_Def_some) G_M
          (ite (= e G_E_M_Def_many) G_M
            (ite (= e G_E_M_Def_seq) G_M
              (ite (= e G_E_M_Def_range) G_M
                (ite (= e G_E_MRange_lb) G_Num
                  (ite (= e G_E_MRange_ub) G_UBound
                    (ite (= e G_E_UBound_Def_Num) G_UBound
                      (ite (= e G_E_UBound_Def_Star) G_UBound
                        (ite (= e G_E_SK_Def_Value) G_SK
                          (ite (= e G_E_SK_Def_Class) G_SK
                            (ite (= e G_E_SDE_Def_C) G_SDE
                              (ite (= e G_E_C_TD) G_TD
                                (ite (= e G_E_C_Id) G_Id

```

```

(ite (= e G_E_SDE_Def_RE)      G_SDE
(ite (= e G_E_RE_Id)           G_Id
(ite (= e G_E_RE_Src_TD)       G_TD
(ite (= e G_E_RE_Tgt_TD)       G_TD
(ite (= e G_E_RE_Src_M)        G_M
(ite (= e G_E_RE_Tgt_M)        G_M
(ite (= e G_E_PED_M)           G_M
(ite (= e G_E_PED_TD)          G_TD
(ite (= e G_E_PED_Id)          G_Id
(ite (= e G_E_SDE_Def_Set)     G_SDE
(ite (= e G_E_Set_Def_PSet)    G_Set
(ite (= e G_E_Set_Def_DSet)    G_Set
(ite (= e G_E_PSet_Id)         G_Id
(ite (= e G_E_PSet_SK)         G_SK
(ite (= e G_E_PSet_isDef)      G_Bool
(ite (= e G_E_PSet-Cs)         G_C
(ite (= e G_E_PSet_PEDs)       G_PED
(ite (= e G_E_PSet_As)         G_A
(ite (= e G_E_PSet_hiOs)       G_O
(ite (= e G_E_PSet_hiPSs)      G_PSet
(ite (= e G_E_DSet_Id)         G_Id
(ite (= e G_E_DSet_SDef)       G_SDef
(ite (= e G_E_SD_SDEs)         G_SDE
(ite (= e G_E_SD_As)           G_A
G_Null))))))))))))))))))))))))))))))))))))))

(define-fun Source_G ((e E_G)) V_G
  (ite (= e G_E_O_Id)          G_O
    (ite (= e G_E_M_Def_opt)    G_M_Opt
      (ite (= e G_E_M_Def_one)  G_M_One
        (ite (= e G_E_M_Def_some) G_M_Some
          (ite (= e G_E_M_Def_many) G_M_Many
            (ite (= e G_E_M_Def_seq) G_M_Seq
              (ite (= e G_E_M_Def_range) G_M_Range
                (ite (= e G_E_MRange_lb) G_M_Range
                  (ite (= e G_E_MRange_ub) G_M_Range
                    (ite (= e G_E_UBound_Def_Num) G_UBound_Num
                      (ite (= e G_E_UBound_Def_Star) G_UBound_Star
                        (ite (= e G_E_SK_Def_Value) G_SK_Value
                          (ite (= e G_E_SK_Def_Class) G_SK_Class
                            (ite (= e G_E_SDE_Def_C) G_C
                              (ite (= e G_E_C_TD) G_C
                                (ite (= e G_E_C_Id) G_C
                                  (ite (= e G_E_SDE_Def_RE) G_RE
                                    (ite (= e G_E_RE_Id) G_RE
                                      (ite (= e G_E_RE_Src_TD) G_RE
                                        (ite (= e G_E_RE_Tgt_TD) G_RE
                                          (ite (= e G_E_RE_Src_M) G_RE
                                            (ite (= e G_E_RE_Tgt_M) G_RE

```

```

(ite (= e G_E_PED_M)          G_PED
(ite (= e G_E_PED_TD)         G_PED
(ite (= e G_E_PED_Id)         G_PED
(ite (= e G_E_SDE_Def_Set)    G_Set
(ite (= e G_E_Set_Def_PSet)   G_PSet
(ite (= e G_E_Set_Def_DSet)   G_DSet
(ite (= e G_E_PSet_Id)        G_PSet
(ite (= e G_E_PSet_SK)        G_PSet
(ite (= e G_E_PSet_isDef)     G_PSet
(ite (= e G_E_PSet-Cs)        G_PSet
(ite (= e G_E_PSet_PEDs)      G_PSet
(ite (= e G_E_PSet_As)        G_PSet
(ite (= e G_E_PSet_hiOs)      G_PSet
(ite (= e G_E_PSet_hiPSs)     G_PSet
(ite (= e G_E_DSet_Id)        G_DSet
(ite (= e G_E_DSet_SDef)      G_DSet
(ite (= e G_E_SD_SDEs)        G_SD
(ite (= e G_E_SD_As)          G_SD
G_Null))))))))))))))))))))))))))))))))))))))

(push)
(echo "Testing function 'Target_MM' (1) --> sat")
(assert (= (Target_MM MM_E_Constant_TD) MM_TypeDesignator))
(check-sat)
(pop)

(push)
(echo "Testing function 'Target_MM' (2)->sat")
(assert (= (Target_MM MM_E_I_RelEdge) MM_SDElem))
(check-sat)
(pop)

(push)
(echo "Testing function 'Target_MM' (3) -> unsat")
(assert (= (Target_MM MM_E_I_SetKind_Value) MM_Num))
(check-sat)
(pop)

(push)
(echo "Checking totality of 'Target_MM' ->sat")
(assert (forall ((emm E_MM))
  (=> (= (Target_MM emm) MM_Null) (= emm MM_ENull))))
(check-sat)
(pop)

(push)
(echo "Checking totality of 'Target_G' ->sat")
(assert (forall ((eg E_G))
  (=> (= (Target_G eg) G_Null) (= eg G_ENull))))

```

```

(check-sat)
(pop)

(push)
(echo "Checking that the target function 'Target_MM' is preserved -> sat")
(assert (forall ((emm1 E_MM))
  (= (Map_V (Target_MM emm1)) (Target_G (Map_E emm1)))))
(check-sat)
(pop)

(push)
(echo "Testing the 'Source_MM' function (1)-> sat")
(assert (= (Source_MM MM_E_RelEdge_Tgt) MM_RelEdge))
(check-sat)
(pop)

(push)
(echo "Testing the 'Source_MM' function (2)-> sat")
(assert (= (Source_MM MM_E_I_SetKind_Value) MM_SetKind_Value))
(check-sat)
(pop)

(push)
(echo "Testing the 'Source_MM' function (3) -> unsat")
(assert (= (Source_MM MM_E_I_SetKind_Class) MM_SetKind_Value))
(check-sat)
(pop)

(push)
(echo "Testing the 'Source_G' function (1)-> sat")
(assert (= (Source_G G_E_RE_Src_M) G_RE))
(check-sat)
(pop)

(push)
(echo "Testing the 'Source_G' function (2) -> sat")
(assert (= (Source_G G_E_SK_Def_Value) G_SK_Value))
(check-sat)
(pop)

(push)
(echo "Testing the 'Source_G' function (3) -> unsat")
(assert (= (Source_G G_E_MRange_lb) G_TD))
(check-sat)
(pop)

(push)
(echo "Checking Totality of 'Source_MM' ->sat")
(assert (forall ((emm E_MM))

```

```

    (=> (= (Source_MM emm) MM_Null) (= emm MM_ENull))))
(check-sat)
(pop)

(push)
(echo "Checking Totality of 'Source_G' ->sat")
(assert (forall ((eg E_G))
    (=> (= (Source_G eg) G_Null) (= eg G_E_Null))))
(check-sat)
(pop)

(push)
(echo "Checking that the source function 'Source_MM' is preserved -> sat")
(assert (forall ((emm1 E_MM))
    (= (Map_V (Source_MM emm1)) (Source_G (Map_E emm1)))))
(check-sat)
(pop)

```

C.2.1 Z3 Output

```

Testing function 'Map_V' (1) --> sat
sat
Testing function 'Map_V' (2) --> sat
sat
Testing function 'Map_V' (3) --> unsat
unsat
Checking Totality of 'Map_V' --> sat
sat
Checking injectiveness of 'Map_V' --> sat
sat
Checking Surjectiveness of 'Map_V' (1) --> sat
sat
Testing function 'Map_E' (1) --> sat
sat
Testing function 'Map_E' (2) --> sat
sat
Testing function 'Map_E' (3) --> unsat
unsat
Checking Totality of 'Map_E' --> sat
sat
Checking injectiveness of 'Map_E' --> sat
sat
Checking Surjectiveness of 'Map_E' (1) --> sat
sat
Testing function 'Target_MM' (1) --> sat
sat
Testing function 'Target_MM' (2)->sat
sat
Testing function 'Target_MM' (3) -> unsat

```

```

unsat
Checking totality of 'Target_MM' ->sat
sat
Checking totality of 'Target_G' ->sat
sat
Checking that the target function 'Target_MM' is preserved -> sat
sat
Testing the 'Source_MM' function (1)-> sat
sat
Testing the 'Source_MM' function (2)-> sat
sat
Testing the 'Source_MM' function (3) -> unsat
unsat
Testing the 'Source_G' function (1)-> sat
sat
Testing the 'Source_G' function (2) -> sat
sat
Testing the 'Source_G' function (3) -> unsat
unsat
Checking Totality of 'Source_MM' ->sat
sat
Checking Totality of 'Source_G' ->sat
sat
Checking that the source function 'Source_MM' is preserved -> sat
sat

```

C.3 Assertion diagrams

```

(set-option :mbqi true)
(set-option :macro-finder true)
(set-option :pull-nested-quantifiers true)
(set-option :produce-unsat-cores true)
(set-option :produce-models true)

(declare-sort V_MM)
(declare-sort E_MM)

(declare-sort V_G)
(declare-sort E_G)

(declare-const MM_Name V_MM)
(declare-const MM_Bool V_MM)
; From 'Common'
(declare-const MM_TypeDesignator V_MM)
(declare-const MM_SetDef V_MM)
(declare-const MM_SetElement V_MM)
(declare-const MM_PropEdgePred V_MM)
(declare-const MM_SetExpression V_MM)

```

```

;; 'FormulaSource'
(declare-const MM_FormulaSource          V_MM)
(declare-const MM_FormulaSourceSet       V_MM)
(declare-const MM_FormulaSourceElem      V_MM)
(declare-const MM_FormulaSourceUnary     V_MM)
(declare-const MM_FormulaSourceSetId     V_MM)
(declare-const MM_FormulaSourceSetDef    V_MM)
(declare-const MM_FormulaSourceUOp       V_MM)
(declare-const MM_FormulaSourceUOp_Card  V_MM)
(declare-const MM_FormulaSourceUOp_Domain V_MM)
(declare-const MM_FormulaSourceUOp_Range V_MM)
(declare-const MM_FormulaSourceUOp_The   V_MM)
;; 'Formula'
(declare-const MM_Formula                V_MM)
(declare-const MM_FormulaNary            V_MM)
(declare-const MM_ArrowsFormula          V_MM)
(declare-const MM_SetFormula             V_MM)
(declare-const MM_FormulaSubset          V_MM)
(declare-const MM_SetFormulaDef          V_MM)
(declare-const MM_SetFormulaShaded       V_MM)
;; 'QFormula'
(declare-const MM_QFormula               V_MM)
(declare-const MM_QDecl                  V_MM)
(declare-const MM_QuantifierKind         V_MM)
(declare-const MM_QuantifierKind_ForAll  V_MM)
(declare-const MM_QuantifierKind_Exists  V_MM)
;; 'FormulaOp'
(declare-const MM_FormulaOp              V_MM)
(declare-const MM_FOp_Implies            V_MM)
(declare-const MM_FOp_And                V_MM)
(declare-const MM_FOp_Or                 V_MM)
(declare-const MM_FOp_Equiv              V_MM)
(declare-const MM_FOp_SeqComp            V_MM)
(declare-const MM_FOp_Not                V_MM)
;; 'Decl'
(declare-const MM_Decl                   V_MM)
(declare-const MM_VarDecl                 V_MM)
(declare-const MM_DeclObj                 V_MM)
(declare-const MM_DeclSet                 V_MM)
(declare-const MM_DeclSeq                 V_MM)
;; 'DeclFormula'
(declare-const MM_DeclFormula             V_MM)
(declare-const MM_DeclFormulaNary         V_MM)
(declare-const MM_DeclFormulaAtom        V_MM)
;; 'Renaming'
(declare-const MM_RenamingExp             V_MM)
;; 'ADiag'
(declare-const MM_ADiag                   V_MM)
;; Special 'Null' constant to check totality

```



```

(declare-const MM_Null          V_MM)

(declare-const G_Id              V_G)
(declare-const G_Bool            V_G)
;; From 'Common'
(declare-const G_TD              V_G)
(declare-const G_SDef            V_G)
(declare-const G_SE              V_G)
(declare-const G_PEP             V_G)
(declare-const G_SExp            V_G)
;; 'AFS'
(declare-const G_AFS             V_G)
(declare-const G_AFS_SE          V_G)
(declare-const G_AFSS            V_G)
(declare-const G_AFS_FSOp        V_G)
(declare-const G_AFSS_SetId      V_G)
(declare-const G_AFSS_SDef       V_G)
(declare-const G_FSOp            V_G)
(declare-const G_FSOp_Card       V_G)
(declare-const G_FSOp_Domain     V_G)
(declare-const G_FSOp_Range      V_G)
(declare-const G_FSOp_The        V_G)
;; 'F'
(declare-const G_F               V_G)
(declare-const G_AF              V_G)
(declare-const G_F_NAry          V_G)
(declare-const G_SF              V_G)
(declare-const G_SF_SDef         V_G)
(declare-const G_SF_shaded       V_G)
(declare-const G_SF_hasIn        V_G)
;; 'QF'
(declare-const G_QF              V_G)
(declare-const G_QD              V_G)
(declare-const G_QK              V_G)
(declare-const G_QK_All          V_G)
(declare-const G_QK_Exists       V_G)
;; 'FOp'
(declare-const G_FOp             V_G)
(declare-const G_FOp_Implies     V_G)
(declare-const G_FOp_Equiv       V_G)
(declare-const G_FOp_And         V_G)
(declare-const G_FOp_Or          V_G)
(declare-const G_FOp_Not         V_G)
(declare-const G_FOp_SeqComp     V_G)
;; 'D'
(declare-const G_D               V_G)
(declare-const G_VD              V_G)
(declare-const G_VD_0            V_G)
(declare-const G_VD_Set          V_G)

```

```

(declare-const G_VD_Seq      V_G)
;; 'DF'
(declare-const G_DF          V_G)
(declare-const G_DFA         V_G)
(declare-const G_DF_NAry     V_G)
;; 'R'
(declare-const G_R           V_G)
;; 'AD'
(declare-const G_AD          V_G)
;; Special 'Null' constant to check totality
(declare-const G_Null        V_G)

;; 'FormulaSource'
(declare-const MM_E_I_FormulaSourceElem      E_MM)
(declare-const MM_E_I_FormulaSourceSet       E_MM)
(declare-const MM_E_I_FormulaSourceUnary     E_MM)
(declare-const MM_E_FormulaSourceUnary_frmlSrc E_MM)
(declare-const MM_E_FormulaSourceUnary_operator E_MM)
(declare-const MM_E_I_FormulaSourceSetId     E_MM)
(declare-const MM_E_FormulaSourceSetId_setId E_MM)
(declare-const MM_E_I_FormulaSourceSetDef    E_MM)
(declare-const MM_E_FormulaSourceSetDef_setDef E_MM)
; 'Formula'
(declare-const MM_E_I_FormulaNAry            E_MM)
(declare-const MM_E_FormulaNAry_frmls       E_MM)
(declare-const MM_E_FormulaNAry_operator    E_MM)
(declare-const MM_E_I_SetFormula            E_MM)
(declare-const MM_E_I_ArrowsFormula         E_MM)
(declare-const MM_E_ArrowsFormula_source    E_MM)
(declare-const MM_E_ArrowsFormula_pes       E_MM)
(declare-const MM_E_I_FormulaSubset         E_MM)
(declare-const MM_E_I_SetFormulaShaded      E_MM)
(declare-const MM_E_I_SetFormulaDef         E_MM)
(declare-const MM_E_FormulaSubset_setId     E_MM)
(declare-const MM_E_FormulaSubset_hasIn     E_MM)
(declare-const MM_E_SetFormulaDef_shaded    E_MM)
(declare-const MM_E_SetFormulaDef_setId     E_MM)
(declare-const MM_E_SetFormulaDef_setDef    E_MM)
(declare-const MM_E_SetFormulaShaded_setId  E_MM)
(declare-const MM_E_I_QFormula              E_MM)
(declare-const MM_E_QFormula_decls          E_MM)
(declare-const MM_E_QFormula_frml          E_MM)
; 'QDecl'
(declare-const MM_E_QDecl_vars              E_MM)
(declare-const MM_E_QDecl_qkind             E_MM)
(declare-const MM_E_I_QuantifierKind_ForAll E_MM)
(declare-const MM_E_I_QuantifierKind_Exists E_MM)
;; 'FormulaOp'
(declare-const MM_E_I_FOp_Implies           E_MM)

```

```

(declare-const MM_E_I_FOp_And E_MM)
(declare-const MM_E_I_FOp_Or E_MM)
(declare-const MM_E_I_FOp_Equiv E_MM)
(declare-const MM_E_I_FOp_SeqComp E_MM)
(declare-const MM_E_I_FOp_Not E_MM)
;; 'Decl'
(declare-const MM_E_I_VarDecl E_MM)
(declare-const MM_E_I_DeclSet E_MM)
(declare-const MM_E_I_DeclSeq E_MM)
(declare-const MM_E_I_DeclObj E_MM)
(declare-const MM_E_DeclObj_optional E_MM)
(declare-const MM_E_VarDecl_dName E_MM)
(declare-const MM_E_VarDecl_dTy E_MM)
(declare-const MM_E_VarDecl_isHidden E_MM)
;; 'DeclFormula'
(declare-const MM_E_I_DeclFormula E_MM)
(declare-const MM_E_I_DeclFormulaNary E_MM)
(declare-const MM_E_DeclFormulaNary_dfop E_MM)
(declare-const MM_E_DeclFormulaNary_dFrmls E_MM)
(declare-const MM_E_I_DeclFormulaAtom E_MM)
(declare-const MM_E_DeclFormulaAtom_refId E_MM)
(declare-const MM_E_DeclFormulaAtom_owningSet E_MM)
(declare-const MM_E_DeclFormulaAtom_callObj E_MM)
(declare-const MM_E_DeclFormulaAtom_import E_MM)
(declare-const MM_E_DeclFormulaAtom_renameExp E_MM)
;; 'Renaming'
(declare-const MM_E_Renaming_subExp E_MM)
(declare-const MM_E_Renaming_varToSub E_MM)
;; 'ADiag'
(declare-const MM_E_ADiag_aName E_MM)
(declare-const MM_E_ADiag_predicate E_MM)
(declare-const MM_E_ADiag_decls E_MM)
;; Special 'Null' constant to check totality
(declare-const MM_E_Null E_MM)

;; 'AFS'
(declare-const G_E_AFS_Def_SE E_G)
(declare-const G_E_AFS_Def_AFSS E_G)
(declare-const G_E_AFS_Def_FSOp E_G)
(declare-const G_E_AFS_FSOp_Op E_G)
(declare-const G_E_AFS_FSOp_AFS E_G)
;; 'AFSS'
(declare-const G_E_AFSS_Def_SetId E_G)
(declare-const G_E_AFSS_Def_SDef E_G)
(declare-const G_E_AFSS_SetId_Id E_G)
(declare-const G_E_AFSS_SDef_SDef E_G)
;; 'F'
(declare-const G_E_F_Def_AF E_G)
(declare-const G_E_F_Def_SF E_G)

```

```

(declare-const G_E_F_Def_NAry      E_G)
(declare-const G_E_F_NAry_Fs       E_G)
(declare-const G_E_F_NAry_FOp      E_G)
(declare-const G_E_AF_AFS           E_G)
(declare-const G_E_AF_PEPs          E_G)
(declare-const G_E_SF_Def_SDef      E_G)
(declare-const G_E_SF_Def_shaded    E_G)
(declare-const G_E_SF_Def_hasIn     E_G)
(declare-const G_E_SF_SDef_shaded   E_G)
(declare-const G_E_SF_SDef_Id       E_G)
(declare-const G_E_SF_SDef_SDef     E_G)
(declare-const G_E_SF_shaded_TD     E_G)
(declare-const G_E_SF_hasIn_TD      E_G)
(declare-const G_E_SF_hasIn_SExp    E_G)
;; 'QF'
(declare-const G_E_F_Def_QF         E_G)
(declare-const G_E_QF_QDs           E_G)
(declare-const G_E_QF_F             E_G)
(declare-const G_E_QD_QK            E_G)
(declare-const G_E_QD_VDs           E_G)
(declare-const G_E_QK_Def_ForAll    E_G)
(declare-const G_E_QK_Def_Exists    E_G)
;; 'FOp'
(declare-const G_E_FOp_Def_Implies  E_G)
(declare-const G_E_FOp_Def_Equiv    E_G)
(declare-const G_E_FOp_Def_And      E_G)
(declare-const G_E_FOp_Def_Or       E_G)
(declare-const G_E_FOp_Def_SeqComp  E_G)
(declare-const G_E_FOp_Def_Not      E_G)
;; 'D'
(declare-const G_E_D_Def_VD         E_G)
(declare-const G_E_D_Def_DF         E_G)
(declare-const G_E_DV_Def_0         E_G)
(declare-const G_E_DV_Def_Set       E_G)
(declare-const G_E_DV_Def_Seq       E_G)
(declare-const G_E_D_VD_Id          E_G)
(declare-const G_E_D_VD_TD          E_G)
(declare-const G_E_D_VD_isHidden    E_G)
(declare-const G_E_DV_Def_0_opt     E_G)
;; 'DF'
(declare-const G_E_DF_Def_DFA       E_G)
(declare-const G_E_DF_Def_NAry      E_G)
(declare-const G_E_DF_NAry_DFs      E_G)
(declare-const G_E_DF_NAry_FOp      E_G)
;; 'DFA'
(declare-const G_E_DFA_uparrow      E_G)
(declare-const G_E_DFA_RefId        E_G)
(declare-const G_E_DFA_ObjId        E_G)
(declare-const G_E_DFA_SetId        E_G)

```

```

(declare-const G_E_DFA_Rs          E_G)
;; 'R'
(declare-const G_E_R_Id1           E_G)
(declare-const G_E_R_Id2           E_G)
;; 'AD'
(declare-const G_E_AD_Id           E_G)
(declare-const G_E_AD_Ds           E_G)
(declare-const G_E_AD_Fs           E_G)
;; Special 'Null' constant to check totality
(declare-const G_E_Null            E_G)

(assert (distinct
  MM_Null
  MM_Name
  MM_Bool
  MM_TypeDesignator
  MM_SetDef
  MM_SetElement
  MM_PropEdgePred
  MM_SetExpression
  MM_FormulaSource
  MM_FormulaSourceSet
  MM_FormulaSourceElem
  MM_FormulaSourceUnary
  MM_FormulaSourceSetId
  MM_FormulaSourceSetDef
  MM_FormulaSourceUOp
  MM_FormulaSourceUOp_Card
  MM_FormulaSourceUOp_Domain
  MM_FormulaSourceUOp_Range
  MM_FormulaSourceUOp_The
  MM_Formula
  MM_ArrowsFormula
  MM_SetFormula
  MM_FormulaNAry
  MM_FormulaSubset
  MM_SetFormulaDef
  MM_SetFormulaShaded
  MM_QFormula
  MM_QDecl
  MM_QuantifierKind
  MM_QuantifierKind_ForAll
  MM_QuantifierKind_Exists
  MM_Decl
  MM_VarDecl
  MM_DeclObj
  MM_DeclSet
  MM_DeclSeq
  MM_DeclFormula

```

```

MM_DeclFormulaNary
MM_DeclFormulaAtom
MM_FormulaOp
MM_FOp_Implies
MM_FOp_And
MM_FOp_Or
MM_FOp_Equiv
MM_FOp_SeqComp
MM_FOp_Not
MM_RenamingExp
MM_ADiag))

(assert (distinct
  G_Null
  G_Id
  G_Bool
  G_TD
  G_SDef
  G_SE
  G_PEP
  G_SExp
  G_AFS
  G_AFS_SE
  G_AFSS
  G_AFS_FSOp
  G_AFSS_SetId
  G_AFSS_SDef
  G_FSOp
  G_FSOp_Card
  G_FSOp_Domain
  G_FSOp_Range
  G_FSOp_The
  G_F
  G_AF
  G_SF
  G_F_NAry
  G_QF
  G_QD
  G_QK
  G_QK_All
  G_QK_Exists
  G_D
  G_VD
  G_VD_0
  G_VD_Set
  G_VD_Seq
  G_DF
  G_DFA
  G_DF_NAry

```

```

G_FOp
G_FOp_Implies
G_FOp_Equiv
G_FOp_And
G_FOp_Or
G_FOp_SeqComp
G_FOp_Not
G_R
G_AD))

(assert (distinct
  MM_E_Null
  MM_E_I_FormulaSourceElem
  MM_E_I_FormulaSourceSet
  MM_E_I_FormulaSourceUnary
  MM_E_FormulaSourceUnary_frmlSrc
  MM_E_FormulaSourceUnary_operator
  MM_E_I_FormulaSourceSetId
  MM_E_FormulaSourceSetId_setId
  MM_E_I_FormulaSourceSetDef
  MM_E_FormulaSourceSetDef_setDef
  MM_E_ArrowsFormula_source
  MM_E_ArrowsFormula_pes
  MM_E_I_FormulaNARY
  MM_E_FormulaNARY_frmls
  MM_E_FormulaNARY_operator
  MM_E_I_FormulaSubset
  MM_E_I_SetFormulaShaded
  MM_E_I_SetFormulaDef
  MM_E_FormulaSubset_setId
  MM_E_FormulaSubset_hasIn
  MM_E_SetFormulaDef_shaded
  MM_E_SetFormulaDef_setId
  MM_E_SetFormulaDef_setDef
  MM_E_SetFormulaShaded_setId
  MM_E_I_QFormula
  MM_E_QFormula_decls
  MM_E_QFormula_frml
  MM_E_QDecl_vars
  MM_E_QDecl_qkind
  MM_E_I_QuantifierKind_ForAll
  MM_E_I_QuantifierKind_Exists
  MM_E_I_VarDecl
  MM_E_I_DeclSet
  MM_E_I_DeclSeq
  MM_E_I_DeclObj
  MM_E_I_DeclFormula
  MM_E_I_DeclFormulaNARY
  MM_E_I_DeclFormulaAtom

```

```

MM_E_DeclFormulaNAry_dfop
MM_E_DeclFormulaNAry_dFrmls
MM_E_DeclFormulaAtom_refId
MM_E_DeclFormulaAtom_import
MM_E_DeclFormulaAtom_renameExp
MM_E_VarDecl_dName
MM_E_VarDecl_dTy
MM_E_VarDecl_isHidden
MM_E_DeclObj_optional
MM_E_I_FOp_Implies
MM_E_I_FOp_And
MM_E_I_FOp_Or
MM_E_I_FOp_Equiv
MM_E_I_FOp_SeqComp
MM_E_I_FOp_Not
MM_E_Renaming_subExp
MM_E_Renaming_varToSub
MM_E_ADiag_aName
MM_E_ADiag_predicate
MM_E_ADiag_decls))

(assert (distinct
  G_E_Null
  G_E_AFS_Def_SE
  G_E_AFS_Def_AFSS
  G_E_AFS_Def_FSOp
  G_E_AFS_FSOp_Op
  G_E_AFS_FSOp_AFS
  G_E_AFSS_Def_SetId
  G_E_AFSS_Def_SDef
  G_E_AFSS_SetId_Id
  G_E_AFSS_SDef_SDef
  G_E_F_Def_AF
  G_E_F_Def_SF
  G_E_F_Def_NAry
  G_E_F_NAry_Fs
  G_E_F_NAry_FOp
  G_E_AF_AFS
  G_E_AF_PEPs
  G_E_SF_Def_SDef
  G_E_SF_Def_shaded
  G_E_SF_Def_hasIn
  G_E_SF_SDef_shaded
  G_E_SF_SDef_Id
  G_E_SF_SDef_SDef
  G_E_SF_shaded_TD
  G_E_SF_hasIn_TD
  G_E_SF_hasIn_SExp
  G_E_F_Def_QF

```



```

G_E_QF_QDs
G_E_QF_F
G_E_QD_QK
G_E_QD_VDs
G_E_QK_Def_ForAll
G_E_QK_Def_Exists
G_E_D_Def_VD
G_E_D_Def_DF
G_E_DV_Def_0
G_E_DV_Def_Set
G_E_DV_Def_Seq
G_E_DF_Def_DFA
G_E_DF_Def_NAry
G_E_D_VD_Id
G_E_D_VD_TD
G_E_D_VD_isHidden
G_E_DV_Def_0_opt
G_E_DF_NAry_DFs
G_E_DF_NAry_FOp
G_E_FOp_Def_Implies
G_E_FOp_Def_Equiv
G_E_FOp_Def_And
G_E_FOp_Def_Or
G_E_FOp_Def_SeqComp
G_E_FOp_Def_Not
G_E_DFA_uparrow
G_E_DFA_RefId
G_E_DFA_ObjId
G_E_DFA_SetId
G_E_DFA_Rs
G_E_R_Id1
G_E_R_Id2
G_E_AD_Id
G_E_AD_Ds
G_E_AD_Fs))

(define-fun Map_V ((v V_MM)) V_G
  (ite (= v MM_Name)          G_Id
    (ite (= v MM_Bool)        G_Bool
      (ite (= v MM_TypeDesignator) G_TD
        (ite (= v MM_SetDef)      G_SDef
          (ite (= v MM_SetElement) G_SE
            (ite (= v MM_PropEdgePred) G_PEP
              (ite (= v MM_SetExpression) G_SExp
                (ite (= v MM_FormulaSource) G_AFS
                  (ite (= v MM_FormulaSourceSet) G_AFSS
                    (ite (= v MM_FormulaSourceElem) G_AFS_SE
                      (ite (= v MM_FormulaSourceUnary) G_AFS_FSOp
                        (ite (= v MM_FormulaSourceSetId) G_AFSS_SetId

```

```

(ite (= v MM_FormulaSourceSetDef)      G_AFSS_SDef
(ite (= v MM_FormulaSourceUOp)        G_FSOp
(ite (= v MM_FormulaSourceUOp_Card)    G_FSOp_Card
(ite (= v MM_FormulaSourceUOp_Domain)  G_FSOp_Domain
(ite (= v MM_FormulaSourceUOp_Range)   G_FSOp_Range
(ite (= v MM_FormulaSourceUOp_The)     G_FSOp_The
(ite (= v MM_Formula)                  G_F
(ite (= v MM_FormulaNAry)              G_F_NAry
(ite (= v MM_SetFormula)                G_SF
(ite (= v MM_ArrowsFormula)            G_AF
(ite (= v MM_FormulaNAry)              G_F_NAry
(ite (= v MM_FormulaSubset)            G_SF_hasIn
(ite (= v MM_SetFormulaDef)            G_SF_SDef
(ite (= v MM_SetFormulaShaded)         G_SF_shaded
(ite (= v MM_QFormula)                 G_QF
(ite (= v MM_QDecl)                    G_QD
(ite (= v MM_QuantifierKind)           G_QK
(ite (= v MM_QuantifierKind_ForAll)    G_QK_All
(ite (= v MM_QuantifierKind_Exists)    G_QK_Exists
(ite (= v MM_Decl)                     G_D
(ite (= v MM_VarDecl)                  G_VD
(ite (= v MM_DeclObj)                  G_VD_0
(ite (= v MM_DeclSet)                  G_VD_Set
(ite (= v MM_DeclSeq)                  G_VD_Seq
(ite (= v MM_DeclFormula)              G_DF
(ite (= v MM_DeclFormulaNAry)          G_DF_NAry
(ite (= v MM_DeclFormulaAtom)          G_DFA
(ite (= v MM_FormulaOp)                G_FOp
(ite (= v MM_FOp_Implies)              G_FOp_Implies
(ite (= v MM_FOp_And)                  G_FOp_And
(ite (= v MM_FOp_Or)                   G_FOp_Or
(ite (= v MM_FOp_Equiv)                G_FOp_Equiv
(ite (= v MM_FOp_SeqComp)              G_FOp_SeqComp
(ite (= v MM_FOp_Not)                  G_FOp_Not
(ite (= v MM_RenamingExp)              G_R
(ite (= v MM_ADiag)                    G_AD
G_Null))))))))))))))))))))))))))))))))))))))

```

```

(define-fun Map_E ((e E_MM)) E_G
  (ite (= e MM_E_I_FormulaSourceElem)      G_E_AFS_Def_SE
  (ite (= e MM_E_I_FormulaSourceSet)        G_E_AFS_Def_AFSS
  (ite (= e MM_E_I_FormulaSourceUnary)      G_E_AFS_Def_FSOp
  (ite (= e MM_E_FormulaSourceUnary_frmlSrc) G_E_AFS_FSOp_AFS
  (ite (= e MM_E_FormulaSourceUnary_operator) G_E_AFS_FSOp_Op
  (ite (= e MM_E_I_FormulaSourceSetId)      G_E_AFSS_Def_SetId
  (ite (= e MM_E_FormulaSourceSetId_setId)  G_E_AFSS_SetId_Id
  (ite (= e MM_E_I_FormulaSourceSetDef)     G_E_AFSS_Def_SDef
  (ite (= e MM_E_FormulaSourceSetDef_setDef) G_E_AFSS_SDef_SDef
  (ite (= e MM_E_I_SetFormula)              G_E_F_Def_SF

```

(ite (= e MM_E_I_ArrowsFormula)	G_E_F_Def_AF
(ite (= e MM_E_I_FormulaNAry)	G_E_F_Def_NAry
(ite (= e MM_E_FormulaNAry_frmls)	G_E_F_NAry_Fs
(ite (= e MM_E_FormulaNAry_operator)	G_E_F_NAry_FOp
(ite (= e MM_E_ArrowsFormula_source)	G_E_AF_AFS
(ite (= e MM_E_ArrowsFormula_pes)	G_E_AF_PEPs
(ite (= e MM_E_I_FormulaSubset)	G_E_SF_Def_hasIn
(ite (= e MM_E_I_SetFormulaShaded)	G_E_SF_Def_shaded
(ite (= e MM_E_I_SetFormulaDef)	G_E_SF_Def_SDef
(ite (= e MM_E_FormulaSubset_setId)	G_E_SF_hasIn_TD
(ite (= e MM_E_FormulaSubset_hasIn)	G_E_SF_hasIn_SExp
(ite (= e MM_E_SetFormulaDef_shaded)	G_E_SF_SDef_shaded
(ite (= e MM_E_SetFormulaDef_setId)	G_E_SF_SDef_Id
(ite (= e MM_E_SetFormulaDef_setDef)	G_E_SF_SDef_SDef
(ite (= e MM_E_SetFormulaShaded_setId)	G_E_SF_shaded_TD
(ite (= e MM_E_I_QFormula)	G_E_F_Def_QF
(ite (= e MM_E_QFormula_decls)	G_E_QF_QDs
(ite (= e MM_E_QFormula_frml)	G_E_QF_F
(ite (= e MM_E_QDecl_vars)	G_E_QD_VDs
(ite (= e MM_E_QDecl_qkind)	G_E_QD_QK
(ite (= e MM_E_I_QuantifierKind_ForAll)	G_E_QK_Def_ForAll
(ite (= e MM_E_I_QuantifierKind_Exists)	G_E_QK_Def_Exists
(ite (= e MM_E_I_VarDecl)	G_E_D_Def_VD
(ite (= e MM_E_I_DeclSet)	G_E_DV_Def_Set
(ite (= e MM_E_I_DeclSeq)	G_E_DV_Def_Seq
(ite (= e MM_E_I_DeclObj)	G_E_DV_Def_0
(ite (= e MM_E_I_DeclFormula)	G_E_D_Def_DF
(ite (= e MM_E_I_DeclFormulaNAry)	G_E_DF_Def_NAry
(ite (= e MM_E_I_DeclFormulaAtom)	G_E_DF_Def_DFA
(ite (= e MM_E_DeclFormulaNAry_dfop)	G_E_DF_NAry_FOp
(ite (= e MM_E_DeclFormulaNAry_dFrmls)	G_E_DF_NAry_DFs
(ite (= e MM_E_VarDecl_dName)	G_E_D_VD_Id
(ite (= e MM_E_VarDecl_dTy)	G_E_D_VD_TD
(ite (= e MM_E_VarDecl_isHidden)	G_E_D_VD_isHidden
(ite (= e MM_E_DeclObj_optional)	G_E_DV_Def_0_opt
(ite (= e MM_E_I_FOp_Implies)	G_E_FOp_Def_Implies
(ite (= e MM_E_I_FOp_And)	G_E_FOp_Def_And
(ite (= e MM_E_I_FOp_Or)	G_E_FOp_Def_Or
(ite (= e MM_E_I_FOp_Equiv)	G_E_FOp_Def_Equiv
(ite (= e MM_E_I_FOp_SeqComp)	G_E_FOp_Def_SeqComp
(ite (= e MM_E_I_FOp_Not)	G_E_FOp_Def_Not
(ite (= e MM_E_DeclFormulaAtom_refId)	G_E_DFA_RefId
(ite (= e MM_E_DeclFormulaAtom_callObj)	G_E_DFA_ObjId
(ite (= e MM_E_DeclFormulaAtom_owningSet)	G_E_DFA_SetId
(ite (= e MM_E_DeclFormulaAtom_import)	G_E_DFA_uparrow
(ite (= e MM_E_DeclFormulaAtom_renameExp)	G_E_DFA_Rs
(ite (= e MM_E_Renaming_subExp)	G_E_R_Id1
(ite (= e MM_E_Renaming_varToSub)	G_E_R_Id2
(ite (= e MM_E_ADiag_aName)	G_E_AD_Id

```

(ite (= e MM_E_ADiag_predicate) G_E_AD_Fs
(ite (= e MM_E_ADiag_decls) G_E_AD_Ds
G_E_Null))))))))))))))))))))))))))))))))))))))))))))))))))))))))))

(push)
(echo "Testing function 'Map_V' (1) --> sat")
(assert (= (Map_V MM_SetDef) G_SDef))
(check-sat)
(pop)

(push)
(echo "Testing function 'Map_V' (2) --> sat")
(assert (= (Map_V MM_PropEdgePred) G_PEP))
(check-sat)
(pop)

(push)
(echo "Testing function 'Map_V' (3) --> unsat")
(assert (= (Map_V MM_FormulaSourceUOp) G_FSOp_Range))
(check-sat)
(pop)

(push)
(echo "Checking Totality of 'Map_V' --> sat")
(assert (forall ((vmm V_MM))
  (=> (= (Map_V vmm) G_Null) (= vmm MM_Null))))
(check-sat)
(pop)

(push)
(echo "Checking injectiveness of 'Map_V' --> sat")
(assert (forall ((vmm1 V_MM) (vmm2 V_MM))
  (=> (= (Map_V vmm1) (Map_V vmm2)) (= vmm1 vmm2))))
(check-sat)
(pop)

(push)
(echo "Checking Surjectiveness of 'Map_V' -->sat")
(assert (forall ((vg V_G))
  (exists ((vmm V_MM))
    (= (Map_V vmm) vg))))
(check-sat)
(pop)

; (push)
; (echo "Checking Surjectiveness of 'Map_V' (2)->sat")
; (declare-fun svmm (V_G) V_MM)
; (assert (forall ((vg V_G))
;   (= (Map_V (svmm vg)) vg)))

```

```

;(check-sat)
;(pop)

(push)
(echo "Testing the 'Map_E' function (1) --> sat")
(assert (= (Map_E MM_E_I_DeclFormulaNAry) G_E_DF_Def_NAry))
(check-sat)
(pop)

(push)
(echo "Testing the 'Map_E' function (2) --> sat")
(assert (= (Map_E MM_E_I_FormulaNAry) G_E_F_Def_NAry))
(check-sat)
(pop)

(push)
(echo "Testing the 'Map_E' function (3) --> unsat")
(assert (= (Map_E MM_E_I_FormulaSourceSetDef) G_E_Null))
(check-sat)
(pop)

(push)
(echo "Checking Totality of 'Map_E' --> sat")
(assert (forall ((emm E_MM))
  (=> (= (Map_E emm) G_E_Null) (= emm MM_E_Null))))
(check-sat)
(pop)

(push)
(echo "Checking injectiveness of 'Map_E' --> sat")
(assert (forall ((emm1 E_MM) (emm2 E_MM))
  (=> (= (Map_E emm1) (Map_E emm2)) (= emm1 emm2))))
(check-sat)
(pop)

(push)
(echo "Checking Surjectiveness of 'Map_E' (1) --> sat")
(assert (forall ((eg E_G))
  (exists ((emm E_MM))
    (= (Map_E emm) eg))))
(check-sat)
(pop)

;(push)
;(echo "Checking surjectiveness of 'Map_E' (2) --> sat")
;(declare-fun semm (E_G) E_MM)
;(assert (forall ((eg E_G))
;  (= (Map_E (semm eg)) eg)))
;(check-sat)

```

; (pop)

```
(define-fun Target_MM ((e E_MM)) V_MM
  (ite (= e MM_E_I_FormulaSourceElem) MM_FormulaSource
    (ite (= e MM_E_I_FormulaSourceSet) MM_FormulaSource
      (ite (= e MM_E_I_FormulaSourceUnary) MM_FormulaSource
        (ite (= e MM_E_FormulaSourceUnary_frmlSrc) MM_FormulaSource
          (ite (= e MM_E_FormulaSourceUnary_operator) MM_FormulaSourceUOp
            (ite (= e MM_E_I_FormulaSourceSetId) MM_FormulaSourceSet
              (ite (= e MM_E_I_FormulaSourceSetDef) MM_FormulaSourceSet
                (ite (= e MM_E_FormulaSourceSetId_setId) MM_Name
                  (ite (= e MM_E_FormulaSourceSetDef_setDef) MM_SetDef
                    (ite (= e MM_E_ArrowsFormula_source) MM_FormulaSource
                      (ite (= e MM_E_ArrowsFormula_pes) MM_PropEdgePred
                        (ite (= e MM_E_I_FormulaNAry) MM_Formula
                          (ite (= e MM_E_FormulaNAry_frmls) MM_Formula
                            (ite (= e MM_E_FormulaNAry_operator) MM_FormulaOp
                              (ite (= e MM_E_I_FormulaSubset) MM_SetFormula
                                (ite (= e MM_E_I_SetFormulaShaded) MM_SetFormula
                                  (ite (= e MM_E_I_SetFormulaDef) MM_SetFormula
                                    (ite (= e MM_E_FormulaSubset_setId) MM_TypeDesignator
                                      (ite (= e MM_E_FormulaSubset_hasIn) MM_SetExpression
                                        (ite (= e MM_E_SetFormulaDef_shaded) MM_Bool
                                          (ite (= e MM_E_SetFormulaDef_setId) MM_Name
                                            (ite (= e MM_E_SetFormulaDef_setDef) MM_SetDef
                                              (ite (= e MM_E_SetFormulaShaded_setId) MM_TypeDesignator
                                                (ite (= e MM_E_I_QFormula) MM_Formula
                                                  (ite (= e MM_E_QFormula_decls) MM_QDecl
                                                    (ite (= e MM_E_QFormula_frml) MM_Formula
                                                      (ite (= e MM_E_QDecl_vars) MM_VarDecl
                                                        (ite (= e MM_E_QDecl_qkind) MM_QuantifierKind
                                                          (ite (= e MM_E_I_QuantifierKind_ForAll) MM_QuantifierKind
                                                            (ite (= e MM_E_I_QuantifierKind_Exists) MM_QuantifierKind
                                                              (ite (= e MM_E_I_VarDecl) MM_Decl
                                                                (ite (= e MM_E_I_DeclSet) MM_VarDecl
                                                                  (ite (= e MM_E_I_DeclObj) MM_VarDecl
                                                                    (ite (= e MM_E_I_DeclSeq) MM_VarDecl
                                                                      (ite (= e MM_E_I_DeclFormula) MM_Decl
                                                                        (ite (= e MM_E_I_DeclFormulaNAry) MM_DeclFormula
                                                                          (ite (= e MM_E_I_DeclFormulaAtom) MM_DeclFormula
                                                                            (ite (= e MM_E_VarDecl_dName) MM_Name
                                                                              (ite (= e MM_E_VarDecl_dTy) MM_TypeDesignator
                                                                                (ite (= e MM_E_VarDecl_isHidden) MM_Bool
                                                                                  (ite (= e MM_E_DeclObj_optional) MM_Bool
                                                                                    (ite (= e MM_E_DeclFormulaNAry_dFrmls) MM_DeclFormula
                                                                                      (ite (= e MM_E_DeclFormulaNAry_dfop) MM_FormulaOp
                                                                                        (ite (= e MM_E_I_FOp_Implies) MM_FormulaOp
                                                                                          (ite (= e MM_E_I_FOp_And) MM_FormulaOp
                                                                                            (ite (= e MM_E_I_FOp_Or) MM_FormulaOp
```

```

(ite (= e MM_E_I_FOp_Equiv) MM_FormulaOp
(ite (= e MM_E_I_FOp_SeqComp) MM_FormulaOp
(ite (= e MM_E_I_FOp_Not) MM_FormulaOp
(ite (= e MM_E_DeclFormulaAtom_refId) MM_Name
(ite (= e MM_E_DeclFormulaAtom_import) MM_Bool
(ite (= e MM_E_DeclFormulaAtom_callObj) MM_Name
(ite (= e MM_E_DeclFormulaAtom_owningSet) MM_Name
(ite (= e MM_E_DeclFormulaAtom_renameExp) MM_RenamingExp
(ite (= e MM_E_Renaming_subExp) MM_Name
(ite (= e MM_E_Renaming_varToSub) MM_Name
(ite (= e MM_E_ADiag_aName) MM_Name
(ite (= e MM_E_ADiag_predicate) MM_Formula
(ite (= e MM_E_ADiag_decls) MM_Decl
MM_Null))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))

(define-fun Source_MM ((e E_MM)) V_MM
  (ite (= e MM_E_I_FormulaSourceElem) MM_FormulaSourceElem
  (ite (= e MM_E_I_FormulaSourceSet) MM_FormulaSourceSet
  (ite (= e MM_E_I_FormulaSourceUnary) MM_FormulaSourceUnary
  (ite (= e MM_E_FormulaSourceUnary_frmlSrc) MM_FormulaSourceUnary
  (ite (= e MM_E_FormulaSourceUnary_operator) MM_FormulaSourceUnary
  (ite (= e MM_E_I_FormulaSourceSetId) MM_FormulaSourceSetId
  (ite (= e MM_E_I_FormulaSourceSetDef) MM_FormulaSourceSetDef
  (ite (= e MM_E_FormulaSourceSetId_setId) MM_FormulaSourceSetId
  (ite (= e MM_E_FormulaSourceSetDef_setDef) MM_FormulaSourceSetDef
  (ite (= e MM_E_ArrowsFormula_source) MM_ArrowsFormula
  (ite (= e MM_E_ArrowsFormula_pes) MM_ArrowsFormula
  (ite (= e MM_E_I_FormulaNary) MM_FormulaNary
  (ite (= e MM_E_FormulaNary_frmls) MM_FormulaNary
  (ite (= e MM_E_FormulaNary_operator) MM_FormulaNary
  (ite (= e MM_E_I_FormulaSubset) MM_FormulaSubset
  (ite (= e MM_E_I_SetFormulaShaded) MM_SetFormulaShaded
  (ite (= e MM_E_I_SetFormulaDef) MM_SetFormulaDef
  (ite (= e MM_E_FormulaSubset_setId) MM_FormulaSubset
  (ite (= e MM_E_FormulaSubset_hasIn) MM_FormulaSubset
  (ite (= e MM_E_SetFormulaDef_shaded) MM_SetFormulaDef
  (ite (= e MM_E_SetFormulaDef_setId) MM_SetFormulaDef
  (ite (= e MM_E_SetFormulaDef_setDef) MM_SetFormulaDef
  (ite (= e MM_E_SetFormulaShaded_setId) MM_SetFormulaShaded
  (ite (= e MM_E_I_QFormula) MM_QFormula
  (ite (= e MM_E_QFormula_decls) MM_QFormula
  (ite (= e MM_E_QFormula_frml) MM_QFormula
  (ite (= e MM_E_QDecl_vars) MM_QDecl
  (ite (= e MM_E_QDecl_qkind) MM_QDecl
  (ite (= e MM_E_I_QuantifierKind_ForAll) MM_QuantifierKind_ForAll
  (ite (= e MM_E_I_QuantifierKind_Exists) MM_QuantifierKind_Exists
  (ite (= e MM_E_I_VarDecl) MM_VarDecl
  (ite (= e MM_E_I_DeclSet) MM_DeclSet
  (ite (= e MM_E_I_DeclObj) MM_DeclObj

```



```

(ite (= e G_E_SF_hasIn_SExp) G_SExp
(ite (= e G_E_SF_SDef_shaded) G_Bool
(ite (= e G_E_SF_SDef_Id) G_Id
(ite (= e G_E_SF_SDef_SDef) G_SDef
(ite (= e G_E_SF_shaded_TD) G_TD
(ite (= e G_E_F_Def_QF) G_F
(ite (= e G_E_QF_QDs) G_QD
(ite (= e G_E_QF_F) G_F
(ite (= e G_E_QD_QK) G_QK
(ite (= e G_E_QD_VDs) G_VD
(ite (= e G_E_QK_Def_ForAll) G_QK
(ite (= e G_E_QK_Def_Exists) G_QK
(ite (= e G_E_D_Def_VD) G_D
(ite (= e G_E_DV_Def_Set) G_VD
(ite (= e G_E_DV_Def_Seq) G_VD
(ite (= e G_E_DV_Def_O) G_VD
(ite (= e G_E_D_Def_DF) G_D
(ite (= e G_E_DF_Def_NAry) G_DF
(ite (= e G_E_DF_Def_DFA) G_DF
(ite (= e G_E_D_VD_Id) G_Id
(ite (= e G_E_D_VD_TD) G_TD
(ite (= e G_E_D_VD_isHidden) G_Bool
(ite (= e G_E_DV_Def_O_opt) G_Bool
(ite (= e G_E_DF_NAry_DFs) G_DF
(ite (= e G_E_DF_NAry_FOp) G_FOp
(ite (= e G_E_FOp_Def_Implies) G_FOp
(ite (= e G_E_FOp_Def_Equiv) G_FOp
(ite (= e G_E_FOp_Def_And) G_FOp
(ite (= e G_E_FOp_Def_Or) G_FOp
(ite (= e G_E_FOp_Def_SeqComp) G_FOp
(ite (= e G_E_FOp_Def_Not) G_FOp
(ite (= e G_E_DFA_RefId) G_Id
(ite (= e G_E_DFA_ObjId) G_Id
(ite (= e G_E_DFA_SetId) G_Id
(ite (= e G_E_DFA_uparrow) G_Bool
(ite (= e G_E_DFA_Rs) G_R
(ite (= e G_E_R_Id1) G_Id
(ite (= e G_E_R_Id2) G_Id
(ite (= e G_E_AD_Id) G_Id
(ite (= e G_E_AD_Fs) G_F
(ite (= e G_E_AD_Ds) G_D
G_Null))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))

(define-fun Source_G ((e E_G)) V_G
  (ite (= e G_E_AFS_Def_SE) G_AFS_SE
    (ite (= e G_E_AFS_Def_AFSS) G_AFSS
      (ite (= e G_E_AFS_Def_FSOp) G_AFS_FSOp
        (ite (= e G_E_AFS_FSOp_Op) G_AFS_FSOp
          (ite (= e G_E_AFS_FSOp_AFS) G_AFS_FSOp

```

(ite (= e G_E_AFSS_Def_SetId)	G_AFSS_SetId
(ite (= e G_E_AFSS_Def_SDef)	G_AFSS_SDef
(ite (= e G_E_AFSS_SetId_Id)	G_AFSS_SetId
(ite (= e G_E_AFSS_SDef_SDef)	G_AFSS_SDef
(ite (= e G_E_AF_AFS)	G_AF
(ite (= e G_E_AF_PEPs)	G_AF
(ite (= e G_E_F_Def_AF)	G_AF
(ite (= e G_E_F_Def_SF)	G_SF
(ite (= e G_E_F_Def_NAry)	G_F_NAry
(ite (= e G_E_F_NAry_Fs)	G_F_NAry
(ite (= e G_E_F_NAry_FOp)	G_F_NAry
(ite (= e G_E_SF_Def_hasIn)	G_SF_hasIn
(ite (= e G_E_SF_Def_shaded)	G_SF_shaded
(ite (= e G_E_SF_Def_SDef)	G_SDef
(ite (= e G_E_SF_hasIn_TD)	G_SF_hasIn
(ite (= e G_E_SF_hasIn_SExp)	G_SF_hasIn
(ite (= e G_E_SF_SDef_shaded)	G_SF_SDef
(ite (= e G_E_SF_SDef_Id)	G_SF_SDef
(ite (= e G_E_SF_SDef_SDef)	G_SF_SDef
(ite (= e G_E_SF_shaded_TD)	G_SF_shaded
(ite (= e G_E_F_Def_QF)	G_QF
(ite (= e G_E_QF_QDs)	G_QF
(ite (= e G_E_QF_F)	G_QF
(ite (= e G_E_QD_QK)	G_QD
(ite (= e G_E_QD_VDs)	G_QD
(ite (= e G_E_QK_Def_ForAll)	G_QK_All
(ite (= e G_E_QK_Def_Exists)	G_QK_Exists
(ite (= e G_E_D_Def_VD)	G_VD
(ite (= e G_E_DV_Def_Set)	G_VD_Set
(ite (= e G_E_DV_Def_Seq)	G_VD_Seq
(ite (= e G_E_DV_Def_O)	G_VD_O
(ite (= e G_E_D_Def_DF)	G_DF
(ite (= e G_E_DF_Def_NAry)	G_DF_NAry
(ite (= e G_E_DF_Def_DFA)	G_DFA
(ite (= e G_E_D_VD_Id)	G_VD
(ite (= e G_E_D_VD_TD)	G_VD
(ite (= e G_E_D_VD_isHidden)	G_VD
(ite (= e G_E_DV_Def_O_opt)	G_VD_O
(ite (= e G_E_DF_NAry_DFs)	G_DF_NAry
(ite (= e G_E_DF_NAry_FOp)	G_DF_NAry
(ite (= e G_E_FOp_Def_Implies)	G_FOp_Implies
(ite (= e G_E_FOp_Def_Equiv)	G_FOp_Equiv
(ite (= e G_E_FOp_Def_And)	G_FOp_And
(ite (= e G_E_FOp_Def_Or)	G_FOp_Or
(ite (= e G_E_FOp_Def_SeqComp)	G_FOp_SeqComp
(ite (= e G_E_FOp_Def_Not)	G_FOp_Not
(ite (= e G_E_DFA_RefId)	G_DFA
(ite (= e G_E_DFA_ObjId)	G_DFA
(ite (= e G_E_DFA_SetId)	G_DFA

```

(ite (= e G_E_DFA_uparrow)      G_DFA
(ite (= e G_E_DFA_Rs)           G_DFA
(ite (= e G_E_R_Id1)            G_R
(ite (= e G_E_R_Id2)            G_R
(ite (= e G_E_AD_Id)            G_AD
(ite (= e G_E_AD_Fs)            G_AD
(ite (= e G_E_AD_Ds)            G_AD
G_Null))))))))))))))))))))))))))))))))))))))))))))))))))))))))))

(push)
(echo "Testing function 'Target_MM' (1) --> sat")
(assert (= (Target_MM MM_E_I_FormulaSourceSet) MM_FormulaSource))
(check-sat)
(pop)

(push)
(echo "Testing function 'Target_MM' (2) --> sat")
(assert (= (Target_MM MM_E_FormulaSourceUnary_operator) MM_FormulaSourceUOp))
(check-sat)
(pop)

(push)
(echo "Testing function 'Target_MM' (3) --> unsat")
(assert (= (Target_MM MM_E_FormulaSourceSetDef_setDef) MM_Null))
(check-sat)
(pop)

(push)
(echo "Checking totality of 'Target_MM' --> sat")
(assert (forall ((emm E_MM))
  (=> (= (Target_MM emm) MM_Null) (= emm MM_E_Null))))
(check-sat)
(pop)

(push)
(echo "Checking totality of 'Target_G' --> sat")
(assert (forall ((eg E_G))
  (=> (= (Target_G eg) G_Null) (= eg G_E_Null))))
(check-sat)
(pop)

(push)
(echo "Checking that the target function 'Target_MM' is preserved --> sat")
(assert (forall ((emm1 E_MM))
  (= (Map_V (Target_MM emm1)) (Target_G (Map_E emm1)))))
(check-sat)
(pop)

(push)

```

```

(echo "Testing function 'Source_MM' (1) --> sat")
(assert (= (Source_MM MM_E_I_FormulaSourceSetId) MM_FormulaSourceSetId))
(check-sat)
(pop)

(push)
(echo "Testing function 'Source_MM' (2) --> sat")
(assert (= (Source_MM MM_E_I_FormulaSourceSetDef) MM_FormulaSourceSetDef))
(check-sat)
(pop)

(push)
(echo "Testing function 'Source_MM' (3) --> unsat")
(assert (= (Source_MM MM_E_FormulaSourceUnary_operator) MM_Null))
(check-sat)
(pop)

(push)
(echo "Checking Totality of 'Source_MM' --> sat")
(assert (forall ((emm E_MM))
  (=> (= (Source_MM emm) MM_Null) (= emm MM_E_Null))))
(check-sat)
(pop)

(push)
(echo "Testing function 'Source_G' (1) --> sat")
(assert (= (Source_G G_E_AFS_FSOp_AFS) G_AFS_FSOp))
(check-sat)
(pop)

(push)
(echo "Testing function 'Source_G' (2) --> sat")
(assert (= (Source_G G_E_AFSS_Def_SetId) G_AFSS_SetId))
(check-sat)
(pop)

(push)
(echo "Testing function 'Source_G' (3) --> unsat")
(assert (= (Source_G G_E_AFSS_SDef_SDef) G_Null))
(check-sat)
(pop)

(push)
(echo "Checking Totality of 'Source_G' -->sat")
(assert (forall ((eg E_G))
  (=> (= (Source_G eg) G_Null) (= eg G_E_Null))))
(check-sat)
(pop)

```

```

(push)
(echo "Checking that the source function 'Source_MM' is preserved --> sat")
(assert (forall ((emm1 E_MM))
  (= (Map_V (Source_MM emm1)) (Source_G (Map_E emm1)))))
(check-sat)
(pop)

```

C.3.1 Z3 Output

```

Testing function 'Map_V' (1) --> sat
sat
Testing function 'Map_V' (2) --> sat
sat
Testing function 'Map_V' (3) --> unsat
unsat
Checking Totality of 'Map_V' --> sat
sat
Checking injectiveness of 'Map_V' --> sat
sat
Checking Surjectiveness of 'Map_V' -->sat
sat
Testing the 'Map_E' function (1) --> sat
sat
Testing the 'Map_E' function (2) --> sat
sat
Testing the 'Map_E' function (3) --> unsat
unsat
Checking Totality of 'Map_E' --> sat
sat
Checking injectiveness of 'Map_E' --> sat
sat
Checking Surjectiveness of 'Map_E' (1) --> sat
sat
Testing function 'Target_MM' (1) --> sat
sat
Testing function 'Target_MM' (2) --> sat
sat
Testing function 'Target_MM' (3) --> unsat
unsat
Checking totality of 'Target_MM' --> sat
sat
Checking totality of 'Target_G' --> sat
sat
Checking that the target function 'Target_MM' is preserved --> sat
sat
Testing function 'Source_MM' (1) --> sat
sat

```

```
Testing function 'Source_MM' (2) --> sat
sat
Testing function 'Source_MM' (3) --> unsat
unsat
Checking Totality of 'Source_MM' --> sat
sat
Testing function 'Source_G' (1) --> sat
sat
Testing function 'Source_G' (2) --> sat
sat
Testing function 'Source_G' (3) --> unsat
unsat
Checking Totality of 'Source_G' -->sat
sat
Checking that the source function 'Source_MM' is preserved --> sat
sat
```